

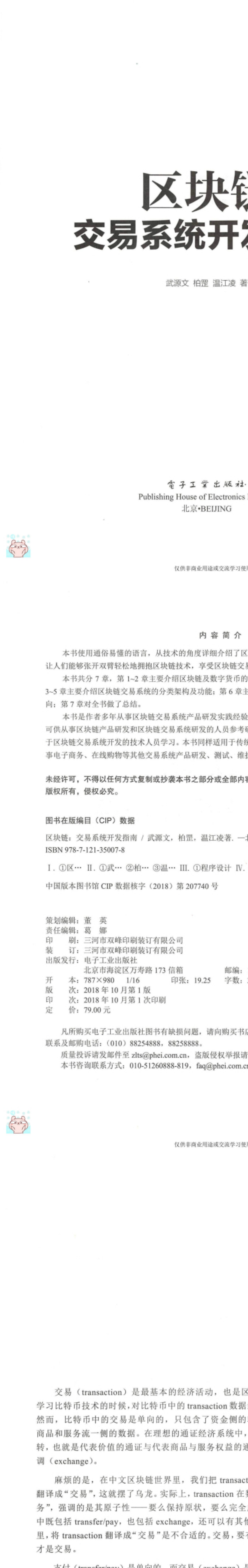
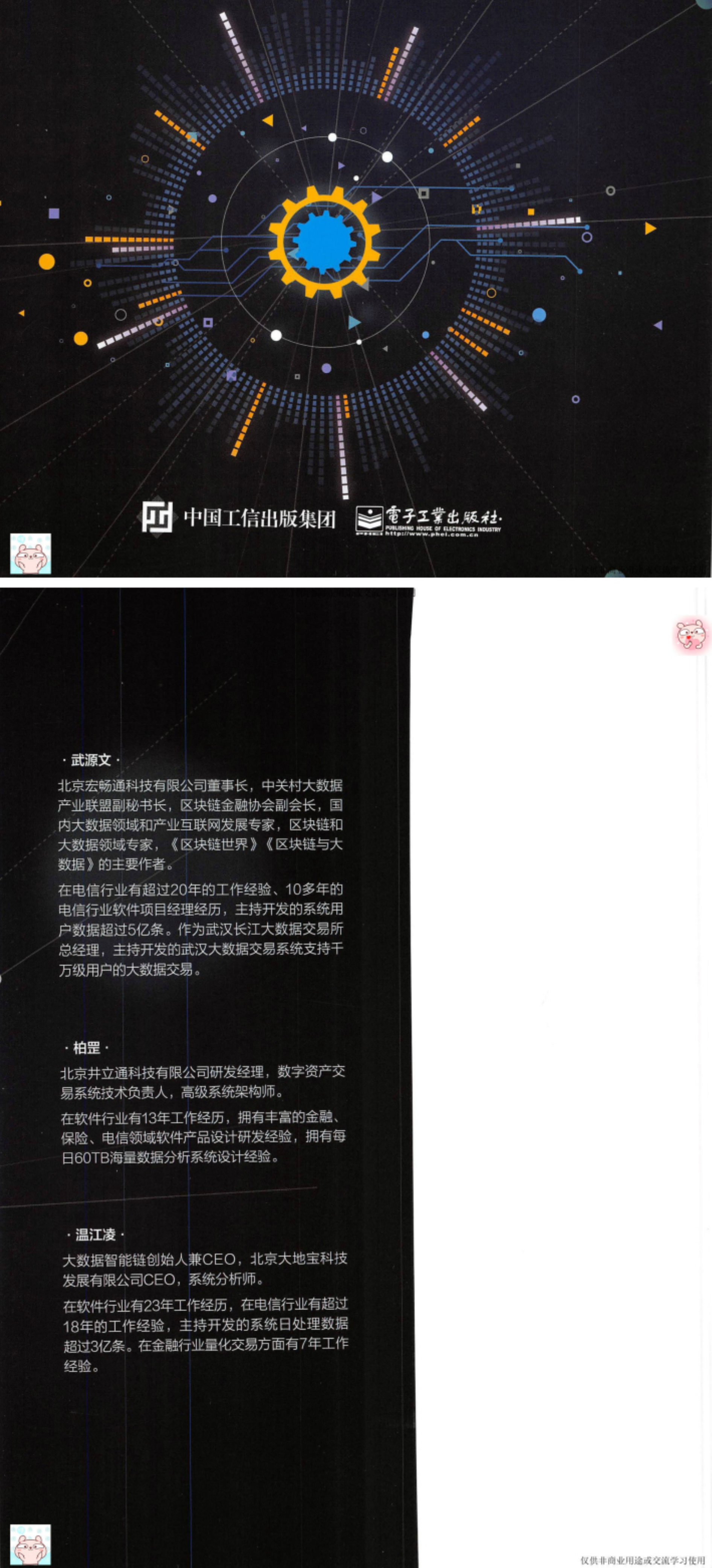


版权相关注意事项：

- 1、书籍版权归著者和出版社所有
- 2、本PDF来自于各个广泛的信息平台，经过整理而成
- 3、本PDF仅限用于非商业用途或者个人交流研究学习使用
- 4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负
- 5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
- 6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
- 7、请于下载PDF后24小时内研究使用并删掉本PDF

版权相关注意事项：

- 1、书籍版权归著者和出版社所有
- 2、本PDF来自于各个广泛的信息平台，经过整理而成
- 3、本PDF仅限用于非商业用途或者个人交流研究学习使用
- 4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负
- 5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
- 6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
- 7、请于下载PDF后24小时内研究使用并删掉本PDF







# 目 录

第 1 章 区块链交易系统基础.....	1
1.1 区块链概述.....	1
1.1.1 区块链的定义.....	1
1.1.2 区块链的核心原理.....	3
1.1.3 区块链的特性.....	4
1.2 区块链分类.....	6
1.2.1 公有链.....	6
1.2.2 私有链.....	7
1.2.3 联盟链.....	7
1.2.4 其他分类方式.....	8
1.3 数字货币.....	8
1.3.1 什么是数字货币.....	8
1.3.2 数字货币与法币的不同.....	8
1.3.3 数字货币的产生和发展.....	9
1.4 数字货币交易.....	11
1.4.1 数字货币交易的特点.....	11
1.4.2 数字货币成交的基本原则.....	11
1.5 区块链交易系统.....	12
1.5.1 区块链交易系统的特点.....	12
1.5.2 区块链交易系统中常见的专业名词.....	13







1.6	本章小结 .....	14
第 2 章	公有链及其 API 接口 .....	15
2.1	BTC .....	15
2.1.1	BTC 公有链的特点 .....	15
2.1.2	BTC 公有链 API 接口 .....	15
2.2	ETH .....	22
2.2.1	ETH 公有链的特点 .....	22
2.2.2	ETH 公有链 API 接口 .....	23
2.3	SWT .....	35
2.3.1	SWT 公有链的特点 .....	35
2.3.2	SWT 公有链 API 接口 .....	35
2.4	MOAC .....	42
2.4.1	MOAC 公有链的特点 .....	42
2.4.2	MOAC 公有链 API 接口 .....	42
2.5	EOS .....	47
2.5.1	EOS 公有链的特点 .....	47
2.5.2	EOS 公有链 API 接口 .....	48
2.6	本章小结 .....	52
第 3 章	交易系统架构 .....	53
3.1	系统概述 .....	53
3.1.1	背景 .....	53
3.1.2	系统目标 .....	54
3.1.3	设计理念 .....	54
3.2	业务功能 .....	60
3.2.1	功能架构 .....	61
3.2.2	功能模块 .....	62
3.2.3	系统流程图 .....	63
3.2.4	业务流程 .....	64
3.3	系统模块 .....	67
3.3.1	服务熔断 .....	67
3.3.2	风控服务 .....	67
3.3.3	数据库设计 .....	68
3.3.4	组网部署结构设计 .....	68
3.4	技术选型 .....	70







3.4.1	ZooKeeper 选型 .....	70
3.4.2	Dubbo 选型 .....	73
3.4.3	中间件选型 .....	81
3.4.4	Redis .....	83
3.4.5	数据库 .....	84
3.4.6	MyBatis .....	87
3.4.7	Druid .....	90
3.4.8	日志收集 .....	91
3.4.9	数据同步 .....	93
3.4.10	数据分析 .....	94
3.4.11	实时计算 .....	95
3.4.12	实时推送 .....	97
3.5	本章小结 .....	98

## 第 4 章 交易系统功能 .....99

4.1	前台功能 .....	99
4.1.1	交易 .....	99
4.1.2	财务中心 .....	118
4.1.3	个人中心 .....	143
4.1.4	服务中心 .....	161
4.2	后台管理概述 .....	164
4.2.1	用户管理 .....	167
4.2.2	交易管理 .....	178
4.2.3	财务管理 .....	211
4.2.4	运营推广 .....	236
4.2.5	系统监控及预警 .....	238
4.3	多语言 .....	249
4.3.1	多语言的目的 .....	249
4.3.2	多语言网站实现方案 .....	250
4.4	软件安全测试 .....	255
4.4.1	安全测试基本概念 .....	255
4.4.2	安全测试的目的 .....	256
4.4.3	安全测试理论 .....	256
4.4.4	安全测试与功能测试的区别 .....	257
4.4.5	安全测试与渗透测试的区别 .....	257
4.4.6	安全测试工具介绍 .....	257
4.5	系统运维 .....	263
4.5.1	平台的数据分类 .....	264







4.5.2	DevOps.....	264
4.5.3	持续集成、持续交付、持续部署 .....	266
4.6	本章小结 .....	277
第 5 章	中心化区块链交易系统.....	278
5.1	中心化区块链交易系统的特点 .....	278
5.1.1	中心化区块链交易系统的机制 .....	278
5.1.2	中心化区块链交易系统的 gas 耗费 .....	280
5.1.3	中心化区块链交易系统的优劣势 .....	281
5.2	去中心化区块链交易系统的特点.....	283
5.2.1	去中心化区块链交易系统的机制 .....	283
5.2.2	去中心化区块链交易系统的 gas 耗费.....	285
5.2.3	去中心化区块链交易系统的优劣势.....	286
5.3	本章小结 .....	287
第 6 章	交易系统的演进 .....	288
6.1	去中心化 .....	288
6.1.1	中心化交易系统 .....	289
6.1.2	去中心化交易系统 .....	292
6.2	证券化 .....	294
6.3	本章小结 .....	295
第 7 章	总结 .....	296
7.1	完美支持各种链 .....	296
7.2	稳定、高可用的系统 .....	298
7.3	交易系统功能齐全.....	298







# 1

## 第 1 章

---

### 区块链交易系统基础

#### 1.1 区块链概述

##### 1.1.1 区块链的定义

区块链技术是构建价值互联网不可或缺的底层应用技术，是具备多级层和多类型应用的价值传输技术的集合。它的本质是一种分布式数据库，或者说是一个可共享且不易更改的分布式分类总账。

该技术方案让参与系统中的任意多个节点，把一段时间系统内的全部信息数据，通过密码学算法计算和记录到一个数据块即区块中，并生成数据“密码”用于验证其信息的有效性和链接下一个数据块，并且由系统的所有参与节点来共同认定记录是否为真。







现在让我们从区块链的起源来更深入地了解区块链。2008 年 11 月 1 日，正当金融危机席卷全球时，一位名叫中本聪的神秘人物向“密码学邮件组”发布了一个帖子：“我们正在开发一种新的电子货币系统，其采用完全点对点的形式，而且无需第三方信托机构。”这样一种不受任何政府或主权控制、去中心化的全球电子货币系统是“密码朋克们”数十年的梦想。

比特币的问世及稳定运行的 10 年证明了区块链技术对于价值传输的可靠性及安全性，开启了互联网由信息互联时代迈向价值互联时代的大门。

在中本聪发布的 *Bitcoin: A Peer-to-Peer Electronic Cash* 论文中，我们看到了这种电子货币体系的几项颠覆式创新。

(1) 去中心化。比特币的发行和流通不依靠中央银行等第三方机构，而是依靠特定算法及密码学技术通过点对点的传输实现，是一种完全依靠网络节点的分布式虚拟货币。

(2) 开源性。在比特币系统中，所有参与者都可以成为比特币的发行者及交易者，整个系统的运作规则是公开透明的，任何个人或机构都可使用比特币系统，整个系统是以开源的方式存在的。

(3) 匿名性。在比特币系统中，任何个人或组织都可以开设比特币账户，而每个账户对应的地址实际上是与用户的现实身份没有任何关系的 ID。比特币持有者可通过不断转换 ID 来隐藏自己的身份。同时，整个比特币网络都不存储可以辨认个人身份的信息。

(4) 不可逆性。全部交易都被加上时间戳，并将交易信息并入一个不断延展的基于散列算法的工作量证明的链条上作为交易记录。除非重新完成全部的工作量证明，否则所形成的交易记录将不可变更。

(5) 安全性。公钥与私钥相结合。公钥用于计算比特币地址，而操控比特币需要私钥，它可以被隔离保存在任何存储介质上，除了用户自己无人可以获取。此外，系统中的每个节点都能获得一份完整数据库的拷贝，得知所有比特币的交易信息。除非同时控制整个系统中超过 51% 的节点，否则在单个节点上对数据库的修改是无效的。因此，比特币的安全性将随着参与者的增加而提升。

(6) 全球自由便捷流通。使用比特币没有烦琐的手续，只需要告知对方比特币地址就可进行支付。任何一台接入互联网的计算机都能被用来管理比特币。

从中本聪的这套点对点电子货币体系中我们可以看到区块链的雏形，即一种不依靠第三方而实现价值转移的分布式账本技术。这种账本具备以下几个特征。







- ◎ 无限扩展性：区块链上的每个区块都可被看作账本中的一页，在区块上记录着一条或多条交易信息，每增加一个区块就相当于账本增加一页，区块链上的区块数量是没有上限的。
- ◎ 全员维护：账本依靠网络中的节点共同记录与维护，不依靠第三方机构。
- ◎ 加密且有序排列：交易信息被加密打包和记录到每一个区块中，并加盖时间戳，一个个区块根据时间戳顺序链接成一个总账本。

在这里，我们必须强调比特币并不等同于区块链，它只是区块链技术的一个早期的最典型的应用范例。这个应用范例的问世打开了区块链的“潘多拉魔盒”，让虚拟的互联网世界开启了价值互联的时代，其核心是依靠技术手段建立一种无需第三方担保的安全可信任的机制，让人人可以参与其中。

### 1.1.2 区块链的核心原理

区块链的核心理念是：构建前后关联且可相互验证的数据块（即区块），并通过时间戳将区块排序，结合密码学技术，形成集体维护、彼此验证、有序链接的网状价值传输系统。

关于区块链，我们需要理解几个核心概念。

#### 1. 区块

在区块链技术中，有价值的信息以数据的形式被永久存储下来，这些用于存储数据信息的载体被称为区块。区块按时间顺序排列，每个区块都记录着它在被创建期间所发生的交易信息，所有区块有序链接起来以汇聚成一本“总账”，而每个区块都可被看作总账中的一页。

每个区块均包含三个要素：①本区块的 ID；②若干交易单；③前一个区块的 ID。

在比特币系统中，每隔 10 分钟创建一个区块，这个区块记录了在这段时间内发生的所有交易信息。同时，每个区块都包含前一个区块的 ID，因此便可根据此 ID 找到上一个区块，依此类推，追踪到起始区块，从而可以生成一个完整的交易链条，形成区块链。

#### 2. 时间戳

顾名思义，时间戳是记录某一事件发生时点的信息。在区块链中从区块生成的那一刻







开始，时间戳便存在于区块中。由于时间的唯一性，让每个加盖了时间戳的区块都是独一无二的，并且提供了认证依据，保证了它的真实性。通过时间戳，各个区块有序排列起来，最后生成一个完整的链条。

### 3. 散列算法

散列算法是区块链中保证交易信息不被篡改的单向密码机制。区块链通过散列算法对一个交易区块中的交易进行加密，并把信息压缩成由一串数字和字母组成的散列字符串。区块链的散列值能够唯一而准确地标识一个区块。在验证区块的真实性时，只需要简单计算出这个区块的散列值，如果没有变化就意味着这个区块上的信息是没有被篡改过的。

### 4. 公钥和私钥

从密码学的角度定义，公钥和私钥其实是一种不对称的加密方式，其核心思想是加密与解密采用不同的密钥。在区块链中使用公钥和私钥标识身份，信息发送者用私钥对信息进行签名，使用信息接收方的公钥对信息加密；信息接收方用信息发送者的公钥验证发送者的身份，使用私钥对加密信息解密。

在介绍了区块链的核心原理和几个核心概念后，我们不难发现，区块链技术是密码学、经济学、分布式存储技术、网络科学及应用数据等多种技术的整合，目的是构建一套可信任的价值传输体系。这些技术按特定规则组合在一起，构建了一套分布式数据记录和存储系统，并通过时间戳为存储数据的区块排序，形成一个连续且前后关联的分布式数据库，这个数据库是价值的天然载体。

## 1.1.3 区块链的特性

与传统记账方式相比，区块链具有去中心化、开放性、自治性、集体维护、信息不可篡改、匿名性、可追溯性、智能性等特性。

### 1. 去中心化

区块链本质上是分布式数据库，因此区块链上的数据发送、验证、存储等均基于分布式系统架构，依靠算法和程序来建立可信任的机制，而非第三方机构。任意节点的权利和义务都是均等的，交易双方可以自证并直接交易，不需要依赖第三方机构的信用背书。同







时，任何一个节点的损坏或者退出都不会影响整个系统的运行。

## 2. 开放性

区块链系统是开放的，除交易各方的私有信息被加密外，区块链的数据对所有人公开，任何人都可以通过公开的接口查询区块链数据和开发相关应用。

## 3. 自治性

区块链采用协商一致的规范和协议（比如一套公开透明的算法），使得整个系统中的所有节点能够在去信任的环境中自由安全地交换数据，使得对“人”的信任改成对机器的信任，任何人为的干预都不起作用。

## 4. 集体维护

区块链系统是由所有参与节点共同维护的系统。区块链上的每一个节点都可以对区块（数据块）进行维护，而整个系统的运行也依赖每一个节点，这是一个人人参与其中的集体维护系统。

## 5. 信息不可篡改

经过验证的信息被上传至区块链后就会被系统永久存储下来，并得到所有参与节点的集体维护。除非能够同时控制系统中超过 51% 的节点，否则在单个节点上对数据库的修改是无效的，因此区块链的数据稳定性和可靠性极高。

## 6. 匿名性

区块链上的信任体系由程序和算法构建，节点之间的交换遵循固定的算法。交易双方无须通过验证现实中的身份信息来让对方产生信任，因此匿名性是区块链很明显的特征。

## 7. 可追溯性

溯源是指追踪记录有形商品或无形信息的流转链条。在区块链上每一个区块都会被加盖时间戳。时间戳既标识了每一个区块独一无二的身份，又让区块实现了有序排列，为信息溯源找到了很好的路径。







## 8. 智能性

在以上 7 个特性的基础上，区块链还具备可编程性、可承载智能合约等技术。这个特性让人们可以根据具体的应用场景，在区块链上创建和部署相关程序，以实现智能化运行。

## 1.2 区块链分类

根据参与者的不同，区块链可以分为公有（Public）链、联盟（Consortium）链和私有（Private）链，如图 1-1 所示。



图 1-1

### 1.2.1 公有链

公有链，顾名思义，任何人都可以参与使用和维护，典型的如比特币区块链，信息是完全公开的。通常公有链也称为非许可链（Permissionless Blockchain），无官方组织及管理机构，无中心服务器，参与的节点按照系统规格自由接入网络，不受控制，节点间基于共识机制开展工作。

公有链是真正意义上的完全去中心化的区块链，它通过密码学保证交易不可篡改，同时也利用密码学验证结合经济上的激励，在互为陌生的网络环境中建立共识，从而形成去中心化的信用机制。公有链中的共识机制一般是工作量证明（PoW）或权益证明（PoS），用户对共识形成的影响力直接取决于他们在网络中拥有资源的占比。

公有链一般适合于数字货币、面向大众的电子商务，以及互联网金融等 B2C、C2C 或 C2B 等应用场景，比特币和以太坊等就是典型的公有链。







## 1.2.2 私有链

私有链，则由集中管理者进行限制，只有内部少数人可以使用，信息不公开。私有链建立在某个企业内部，系统的运作规则根据企业要求进行设定。

私有链的应用场景一般是企业内部的应用，如数据库管理、审计等；在政府行业也会有一些应用，比如政府的预算和执行，或者政府的行业统计数据等，一般由政府登记，但公众有权力监督。私有链的价值主要是提供安全的、可追溯的、不可篡改的、自动执行的运算平台，可以同时防范来自内部和外部对数据的安全攻击，这在传统的系统中是很难做到的。

## 1.2.3 联盟链

联盟链则介于公有链和私有链之间，是指由若干组织一起合作维护的区块链，使用该区块链必须有权限，相关信息会得到保护，典型的如银联组织等。

联盟链是一种需要注册许可的区块链，这种区块链也称为许可链（Permissioned Blockchain）。联盟链仅限于联盟成员参与，区块链上的读写权限、参与记账权限按照联盟规则来设定。整个网络由成员机构共同维护，网络一般通过成员机构的网关节点接入，共识过程由预先选好的节点控制。由于参与共识的节点比较少，联盟链一般不采用工作量证明的挖矿机制，而是多采用权益证明（PoS）或 PBFT（Practical Byzantine Fault Tolerant）、RAFT 等共识算法。

一般来说，联盟链适合于机构间的交易、结算或清算等 B2B 场景。例如在银行间进行支付、结算、清算的系统就可以采用联盟链的形式，将各家银行的网关节点作为记账节点，如果网络上有超过 2/3 的节点确认一个区块，该区块上记录的交易就将得到全网确认。联盟链对交易的确认时间、每秒交易数都与公有链有较大的区别，对安全和性能的要求也比公有链高。

由 40 多家银行参与的区块链联盟 R3 和 Linux 基金会支持的超级账本（Hyperledger）项目都属于联盟链架构。目前国内有影响力的区块链联盟——中国分布式总账基础协议联盟（ChinaLedger）、中国区块链研究联盟、金链盟（金融区块链联盟）等也都在致力于开发联盟区块链项目。

目前来看，公有链将会更多地吸引社区和媒体的眼球，但更多的商业价值应该体现在







联盟链和私有链上。

### 1.2.4 其他分类方式

根据使用目的和场景的不同，区块链又可以分为以数字货币为目的的货币链、以记录产权为目的的产权链、以众筹为目的的众筹链等。

## 1.3 数字货币

### 1.3.1 什么是数字货币

数字货币是一种匿名性的虚拟货币，不依靠法定货币机构发行，不受央行管控。它依据全世界计算机运算的一组方程式开源代码，通过计算机显卡、CPU 大量的运算处理产生，并使用密码学的设计来确保货币流通各个环节的安全性。基于密码学的设计可以使数字货币只能被真实的拥有者转移或支付。

数字货币与区块链是有机结合在一起的，是紧密相连的关系，区块链是数字货币的最底层技术，也是最重要的技术手段。区块链最成功的实践是在货币领域的创新，即作为数字货币的技术之一。数字货币的使用技术还包括移动支付、可信可控云计算、密码算法等，而比特币的风靡让人们知道了区块链的技术框架及广阔的应用前景。

区块链其实就是一种新兴的数字记账簿，这种账簿拥有强大的功能，相当于一种云存储功能，每完成一定时段的交易后，就把该时段内的所有交易记录下来，且在所有的节点上都进行完整拷贝，这就是一个“区块”。因此，信息几乎没有被篡改的可能，除非有办法入侵几乎所有的节点。一个个区块首尾相连，就构成了区块链。

综上所述，数字货币就是一种加密货币的形式所在。正是因为这种数字货币需要以加密形式存在，所以需要区块链技术支持。区块链技术也是世界上最先进的一种技术，世界上很多知名的企业正在研究这种技术，该技术的发展前景是不可限量的。

### 1.3.2 数字货币与法币的不同

数字货币总量有限，具有极强的数量稀缺性。因为这一组方程式开源代码总量是有限







的，必须通过计算机显卡、CPU 的运算才可以获得。所以数字货币开采得越多，其价值越高。

法币由政府发行，政府是一个中心机构，只要它想，其实是可以任意滥发法币的，就像津巴布韦币那样，可以导致无限制的通货膨胀。而数字货币以数学和程序的形式并且以去中心化的方式发行，有固定的币总量和独特的发行方式。因为绝大部分人并不了解法币是从哪里来的，只是认为法币是自己凭劳动赚来的，根本不会去考虑其实质是国家印出来的纸。它本身不具有多少价值，政府以不具有多少价值的东西，最后换取了你有价值的劳动，靠的正是所谓的“法偿性和强制性”。这也正是有些政府指责数字货币的关键理由，它认为数字货币不具有“法偿性和强制性”，不具有货币属性，不能算货币，更不能以其计价。

然而，什么是法偿性和强制性呢？就是政府以法律的形式强制人民使用法币，也因此叫作法定货币，简称法币。而数字货币却没有强迫任何人使用，使用者完全出于自愿。另外，数字货币是人人都可以发行的货币，只要能让其他人相信你发行的货币具有交换价值就能进入流通环节，当然你发行的货币还有可能会因为得不到信任而价值归零。

### 1.3.3 数字货币的产生和发展

数字货币不同于虚拟世界中的虚拟货币，因为它能被用于真实的商品和服务交易，而不局限在网络游戏中。目前全世界发行的数字货币有数百种。

数字货币的发展历程如下：

2008 年，中本聪在 metzdowd 的密码学邮件组列表中发表了一篇文章，论文描述了比特币的电子现金系统。

2009 年 1 月 3 日，比特币网络诞生，中本聪本人发布了开源的第一版比特币客户端，50 个比特币问世。

2009 年 10 月，首个比特币汇率公布：1 美元兑换 1309.03 个比特币。

2010 年 5 月 21 日，一个程序员用 10000 个比特币购买了一个比萨，第一笔交易产生。

2010 年 7 月 11 日，比特币新版客户端消息被著名新闻网站 Slashdot 提及，为比特币带来大量新用户。

2010 年 7 月 16 日，经过为期 5 天的 10 倍暴涨，BTC 价格从 0.008 美元升值到 0.08







美元，第一次价格的剧烈波动，显示新生事物的崛起。

2010 年 7 月 17 日，第一个比特币平台成立。

2011 年，维基解密、自由网、Singularity Institute、互联网档案馆、自由软件基金会以及其他一些组织，开始接受比特币的捐赠。

2011 年 1 月 27 日，最大数字的比特币交易产生：三个来自津巴布韦的账单在 Bitcoin-otc 上以每 4 个 BTC 换 100 万亿津元。

2011 年 2 月 9 日，BTC 价格首次达 1 美元，与美元等价。

2011 年 5 月 29 日，瑞典海盗的创始人 Rickard Falkvinge 宣布将自己所有的财产都换成了 BTC，此外他还借了很多钱大量囤积 BTC。

2012 年 10 月，BitPay 发布报告说，有超过 1000 家商户通过其支付系统来接收比特币的付款。

2012 年 11 月，WordPress 宣布接受比特币付款。声明说肯尼亚、海地和古巴等地区遭受国际支付系统的封锁，比特币可以帮助这些地区的互联网用户购买服务。

2012 年 12 月 6 日，首家在欧盟法律框架下进行运作的比特币交易所——法国比特币中央交易所诞生，这是世界首家官方认可的比特币交易所。

2013 年 3 月 16 日，人口仅 110 万的欧盟成员国塞浦路斯政府冻结民众银行转账交易，对银行存款账户征税，18 日关闭银行和股市。此时比特币价格为 47.45 美元，接着发生了在比特币历史上绝无仅有的又一次爆炸，经过不到一个月的时间，比特币价格突破了 266 美元大关。

2013 年 4 月 1 日，比特币价格在所有交易市场超过 100 美元。

2013 年 4 月，海盗湾、EZTV 开始接受比特币捐款。

2013 年 4 月，中国四川省雅安地震后，公募基金壹基金宣布接受比特币作为地震捐款。

2013 年 8 月 8 日，比特币被美国德州联邦法官 Hirsh 裁为合法货币。

2013 年 6 月底，德国议会决定持有比特币一年以上将予以免税，随后比特币被德国财政部认定为“记账单位”，这意味着比特币在德国已被视为合法货币，并且可以用来交税和从事贸易活动。







2013 年 8 月 19 日，德国政府认可了比特币的法律和税收地位，成为全球第一个正式认可比特币合法身份的国家。

2013 年 10 月 29 日，全球第一个比特币自动提款机在加拿大温哥华激活，但交易限额为每天 3000 加元。

2013 年 11 月 20 日，美国司法部和美国证交会的代表在北京时间周一晚间出席美国参议院的一个听证会时称，比特币是一种合法的金融工具，这一说法将会推进比特币合法化的进程。

2014 年 5 月 30 日，以色列将成为首个“无钞国家”，实体货币终将消失。

2015 年 3 月 5 日，英格兰银行计划发行一种数字货币。

2015 年 4 月 30 日，高盛加入挖掘数字货币技术潜力的行列。

2015 年 7 月 10 日，花旗银行承认正在开发自己的数字货币——花旗币（CitiCoin）。

2015 年 9 月 8 日，瑞士银行利用比特币技术开发虚拟货币。

2020 年，挪威将步入无现金时代。

## 1.4 数字货币交易

### 1.4.1 数字货币交易的特点

- ◎ 交易时间：7×24 小时，全年无休市。
- ◎ 无涨跌停：数字货币交易无涨跌停限制，例如 5 月 28 日比特币单日涨幅超 20%。
- ◎ 交易单位：最小可到小数点后 4 位，没有股票最少买一手（100 股）的买入限制。
- ◎ 随时交易：数字货币是 T+0 交易，当天买入当天即可卖出。

### 1.4.2 数字货币成交的基本原则

- ◎ 限价交易：投资者可以设定低于市场价格的买入价格或高于市场价格的卖出价格的委托，当市场价格波动到其设定的价格时即成交。当设定的价格和市价偏离较







大时，容易出现无法成交的结果。

- ◎ 市价交易：以当时的市场价成交，在一定程度上可以保证投资者买卖指令及时成交，但是在市价委托下单前投资者无法预知其交易价格，存在一定的不确定性。一般来说，行情波动越剧烈，市价交易的成交价格的不确定性风险越大。
- ◎ 成交的基本原则：“价格优先，时间优先”原则。较高的买入价格优于较低的买入价格成交，较低的卖出价格优于较高的卖出价格成交，当委托价格一样时，挂单时间较早的委托单优于挂单时间较晚的委托单成交。

## 1.5 区块链交易系统

### 1.5.1 区块链交易系统的特点

区块链交易系统是进行数字资产交换、流通的系统。区块链交易系统以“让数字资产交易更安全、更快捷”为导向，旨在打造一个安全、稳定、快捷、透明的交易系统。

区块链交易系统通过收取交易手续费、项目上市费等方式盈利，交易模式主要为币币交易和场外交易。交易所的主要特点如下：

- ◎ 安全性高（资金安全 and 信息安全）。
- ◎ 平台流动性好。
- ◎ 交易费用低。
- ◎ 交易速度快，用户体验好。
- ◎ 没有提币资金限制。
- ◎ 支持多种衍生品。

区块链交易系统为持币用户提供了安全可靠的交易空间。与传统的金融交易系统（如证券交易所等）不同，区块链交易系统基于区块链分布式账本技术，天生就具有去中心化、防篡改、防丢失、公开透明的特性。





### 1.5.2 区块链交易系统中常见的专业名词

- ◎ 仓位：投资人实有投资和实际投资资金的比例。
- ◎ 全仓：将所有资金一次性全部买入数字货币。
- ◎ 减仓：卖出部分数字货币。
- ◎ 重仓：可用资金和数字货币相比，数字货币份额占多。
- ◎ 轻仓：可用资金和数字货币相比，可用资金份额占多。
- ◎ 空仓：把手里所持数字货币全部卖出，全部转换为资金。
- ◎ 止盈：当获得一定收益后，将所持数字货币卖出以保住盈利。
- ◎ 止损：当亏损到一定程度后，将所持数字货币卖出，以防止亏损进一步扩大。
- ◎ 牛市：价格持续上升，前景乐观。
- ◎ 熊市：价格持续下跌，前景黯淡。
- ◎ 多头（做多）：买方，认为币价未来会上涨，买入币，待币价上涨后，高价卖出获利。
- ◎ 空头（做空）：卖方，认为币价未来会下跌，将手中持有的币（或向交易平台借币）卖出，待币价下跌后，低价买入获利。
- ◎ 建仓：买入数字货币。
- ◎ 补仓：分批买入数字货币，如先买入 1 BTC，之后再买入 1 BTC。
- ◎ 反弹：当币价下跌时，因下跌过快而价格回升调整。
- ◎ 盘整（横盘）：价格波动幅度较小，币价稳定。
- ◎ 跌：币价缓慢下滑。
- ◎ 跳水（瀑布）：币价快速下跌，幅度很大。
- ◎ 割肉：在买入数字货币后，币价下跌，为避免亏损扩大而赔本卖出数字货币；或者当借币做空后，币价上涨，赔本买入数字货币。
- ◎ 套牢：预期币价上涨，不料买入后币价却下跌；或者预期币价下跌，不料卖出后







币价却上涨。

- ◎ 解套：在买入数字货币后，币价下跌造成暂时的账面损失，但之后币价回升，扭亏为盈。
- ◎ 踏空：因看淡后市卖出数字货币后，币价却一路上涨，未能及时买入，因此未能赚得利润。
- ◎ 超买：币价持续上升到一定高度，买方力量基本用尽，币价即将下跌。
- ◎ 超卖：币价持续下跌到一定低点，卖方力量基本用尽，币价即将回升。
- ◎ 诱多：币价盘整已久，下跌可能性较大，空头大多已卖出数字货币，突然空方将币价拉高，诱使多方以为币价将会上涨，纷纷买入，结果空方打压币价，使多方套牢。
- ◎ 诱空：多头买入数字货币后，故意打压币价，使空头以为币价将会下跌，纷纷抛出，结果误入多头的陷阱。

## 1.6 本章小结

本章主要介绍了区块链的定义、区块链的核心原理和特性、区块链分类、数字货币、数字货币交易、区块链交易系统等内容。要了解区块链交易系统，必须先清楚以下几个方面：区块链上的数字货币是如何产生的、数字货币交易的特点、数字货币成交的基本原则、区块链交易系统的特点、区块链交易系统中一些常见的专业名词。本章为读者阅读后续章节提供了一些知识积累，可以帮助其更好地学习后面的内容。





# 2

## 第 2 章

---

### 公有链及其 API 接口

#### 2.1 BTC

---

##### 2.1.1 BTC公有链的特点

---

在最早的区块链 1.0 时代，没有账户的概念，用户余额是从各自在区块链上所有未花费交易输出（UTXO）计算得来的。采用 PoW 共识机制，依赖机器进行哈希运算来获取记账权，无法避免矿池算力集中的问题。

##### 2.1.2 BTC公有链API接口

---

BTC 公有链的主要接口有创建钱包账户、查询账户余额、交易、查询交易信息、查询







当前区块交易记录、查询区块信息。下面通过 JSONRpc 请求方式进行讲解。

### 1. 创建钱包账户

接口：`http:// + access_key + ":" + secret_key + "@" + ip + ":" + port` POST

接口参数如表 2-1 所示。

表 2-1

参数	类型	说明
access_key	String	钱包服务器用户名
secret_key	String	钱包服务器密码

提交参数详情，如表 2-2 所示。

表 2-2

参数	类型	说明
method	String	方法名
params	String	[注册用户名]

提交的数据如下：

```
{
  "method": "getnewaddress",
  "params": "[weiqingwei@126.com]"
}
```

结果如下：

```
{
  "result": "1D49jJv3o2xvVv4DLxV9N7tEPUY4RcYDTy",
}
```

返回的结果信息如表 2-3 所示。

表 2-3

参数	类型	说明
result	String	BTC 钱包地址





## 2. 查询账户余额

接口：`http:// + username + ":" + pass + "@" + ip + ":" + port` POST

接口参数如表 2-4 所示。

表 2-4

参数	类型	说明
Username	String	钱包用户名
Pass	String	钱包密码

提交参数详情，如表 2-5 所示。

表 2-5

参数	类型	说明
method	String	方法名
params	String	[]

提交的数据如下：

```
{
  "method": "getbalance",
  "params": "[]"
}
```

结果如下：

```
{
  "result": "1.000000000000",
}
```

返回的结果信息如表 2-6 所示。

表 2-6

参数	类型	说明
result	String	余额数值

## 3. 交易

接口：`http:// + ":" + pass + "@" + ip + ":" + port` POST







接口参数如表 2-7 所示。

表 2-7

参数	类型	说明
username	String	钱包用户名
pass	String	钱包密码

提交参数详情，如表 2-8 所示。

表 2-8

参数	类型	说明
method	String	方法名
params	String	条件参数[转出地址，转出数量]

提交的数据如下：

```
{
  "method": "sendtoaddress",
  "params": "[\"1D49jJv3o2xvVv4DLxV9N7tEPUY4RcYDTy\",1.0000000000]"
}
```

结果如下：

```
{
  "result": "1C3111148F01A21A5BA1B50E30D716ED1CAB4F7C5
1A36D1D106590B351A5E77C"
}
```

返回的结果信息如表 2-9 所示。

表 2-9

参数	类型	说明
result	String	交易后的 Hash 值

#### 4. 查询交易信息

接口：http://ip + ":" + port POST

提交参数详情，如表 2-10 所示。





表 2-10

参数	类型	说明
method	String	方法名
params	String	条件参数[交易 Hash]

提交的数据如下:

```
{
  "method": "gettransaction",
  "params": "[1C3111148F01A21A5BA1B50E30D716ED1CAB4F7C5
1A36D1D106590B351A5E77C]"
}
```

结果如下:

```
{
  "amount": 0,
  "fee": 0,
  "blockindex": 179123,
  "details": [{
    "fee": 0,
    "amount": 0.01000000,
    "blockindex": 179123,
    "category": "receive",
    "confirmations": 0,
    "address": "1APsgU92VV77GB2YMNNMMYz7Sock5gMgV1",
    "txid": "5031738bc1f797e5e0f8b782989111d751064
96c5dedea50d96e2ed1dc88190d",
    "block": 1327599863,
    "blockhash": "000000000000079bae4b877ad3810f03db249
a6f239c2b69c18d44c141c470ee"
  },
  {
    "fee": 0,
    "amount": 0.00100000,
    "blockindex": 179123,
    "category": "send",
    "confirmations": 0,
    "address": "15CDCKBLsvX3nZ3krMYNse6FkRcuMD1rmU",
    "txid": "5031738bc1f797e5e0f8b782989111d75106
496c5dedea50d96e2ed1dc88190d",
```







```
        "block": 1327599863,
        "blockhash": "000000000000079bae4b877ad3810f03d
b249a6f239c2b69c18d44c141c470ee"
    }
],
"confirmations": 15767,
"txid": "5031738bc1f797e5e0f8b782989111d751
06496c5dedea50d96e2ed1dc88190d",
"block": 1327599863,
"blockhash": "000000000000079bae4b877ad3810f03d
b249a6f239c2b69c18d44c141c470ee"
}
```

返回的结果信息如表 2-11 所示。

表 2-11

参数	类型	说明
txid	String	交易 ID
confirmations	Long	确认该交易的区块数量
block	Long	交易所在的区块
category	String	固定值为交易的付款方和收款方
amount	BigDecimal	交易金额
fee	BigDecimal	交易费用

## 5. 查询当前区块的交易记录

接口：http://ip + ":" + port POST

提交参数详情，如表 2-12 所示。

表 2-12

参数	类型	说明
method	String	方法名
params	String	条件参数[区块 Hash]

提交的数据如下：

```
{
    "method": "gettransaction",
```





```

    "params": "[00000000000003438e8c67500f34dd32bb1b
f0d251a5c230c407641961c85b41]"
  }

```

结果如下：

```

{
  "lastblock": "00000000000009133d70c6282279bfc5fadfea07
e27543445a199fe6ef84b51b",
  "transactions": [{
    "fee": 0.01000000,
    "amount": 1,
    "blockindex": 171984,
    "category": "receive",
    "confirmations": 0,
    "address": "1A8JiWcwpvY7tAopUkSnGuEYHmzGYfZPiq",
    "txid": "90992bc8ebff9774cfc91738863602010dab
9ab2f5b0841cc4922786a2029725",
    "block": 1323486876,
    "blockhash": "00000000000003438e8c67500f34dd32bb1b
f0d251a5c230c407641961c85b41",
    "account": "My Wallet"
  }]
}

```

返回的结果信息：类似于交易接口，这里不再赘述。

## 6. 查询区块信息

接口：http://ip + ":" + port POST

提交参数详情，如表 2-13 所示。

表 2-13

参数	类型	说明
method	String	方法名
params	String	条件参数[区块 Hash]

提交的数据如下：

```

{
  "tx": [

```







```
      "4a5e1e4baab89f3a32518a88c31bc87f618f7667
3e2cc77ab2127b7afdeda33b"
    ],
    "time": 1231006505,
    "height": 0,
    "nonce": 2083236893,
    "hash": "000000000019d6689c085ae165831e934ff76
3ae46a2a6c172b3f1b60a8ce26f",
    "bits": 486604799,
    "difficulty": 1,
    "merkleroot": "4a5e1e4baab89f3a32518a88c31bc87f618f7667
3e2cc77ab2127b7afdeda33b",
    "version": 1,
    "size": 285
  }
```

返回的结果信息如表 2-14 所示。

表 2-14

参数	类型	说明
difficulty	String	区块的难度值，整数
version	Long	版本
size	Long	区块中的交易数量
hash	String	区块 Hash
nonce	BigDecimal	生成的工作量证明的散列值

## 2.2 ETH

### 2.2.1 ETH公有链的特点

以太坊可以被解释为区块链+智能合约。其具备图灵完备性，支持智能合约，可以实现各种商业与非商业环境下的复杂逻辑，隐藏了底层技术的复杂性，让应用开发者更多地专注在应用逻辑及商业逻辑上。以太坊的不足之处在于其扩展性较差，和比特币一样遭受着每笔交易都需要网络中的每个节点来处理这一困境的折磨。2000TPS 的交易就可能导致以太坊因链上的存储快速增长而拥堵。随着接入应用的增多，后期可能更加拥堵。好在以





以太坊全节点只需存储状态而不是完整的区块链。

## 2.2.2 ETH公有链API接口

ETH 公有链的主要接口有创建钱包账户、查询账户余额、解锁账户、交易、查询交易信息、查询最新区块高度、查询当前区块交易总数、查询协议版本、查询预测 gas 值。下面通过 JSONRpc 请求方式进行讲解。

### 1. 创建钱包账户

请求路径：`http://ip+":"+port POST`

提交参数详情，如表 2-15 所示。

表 2-15

参数	类型	说明
jsonrpc	String	版本号
id	String	
method	String	方法名
params	String[]	可变参数数组[密钥, ...]

提交的数据如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "method": "personal_newAccount",
  "params": ["f2fa1d6ae2fc4f3a9ca58cdb6153488"]
}
```

结果如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "result": "0x63500d26c7a3fcee1e43e5b3e9a72adf0bd3c376a",
}
```

返回的结果信息如表 2-16 所示。







表 2-16

参数	类型	说明
jsonrpc	String	版本号
id	String	
result	String	生成的以太坊钱包账户地址

## 2. 查询账户余额（以太坊主币和代币，请求方式不同）

请求路径：`http://ip+":"+port` POST

（1）提交参数详情（主币），如表 2-17 所示。

表 2-17

参数	类型	说明
jsonrpc	String	版本号
id	String	
method	String	方法名
params	String[]	可变参数数组[地址, ...]

提交的数据如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "method": "eth_getBalance",
  "params": ["0x63500d26c7a3fce1e43e5b3e9a72adf0bd3c376a", "latest"]
}
```

结果如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "result": "0x70c6b6d42a8f600",
}
```

返回的结果信息如表 2-18 所示。





表 2-18

参数	类型	说明
jsonrpc	String	版本号
id	String	
result	String	十六进制形式的余额（需要除以以太坊的 unit）

（2）提交参数详情（代币），如表 2-19 所示。

表 2-19

参数	类型	说明
jsonrpc	String	版本号
id	String	
method	String	方法名
params	Object[]	可变参数数组[合约地址,...]，参数不同于主币

提交的数据如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "method": "eth_call",
  "params": [{
    "to": "0x1c6890825880566dd6ad88147e0a6ace7930b7c0",
    "data": "0x70a0823100000000000000000000000000000000752ebe2
2bdef84e5c2f6b4c918c637652fdfe2e"
  },
  "latest"
]
```

结果同上。

### 3. 解锁账户

请求路径：http://ip+":port POST

提交参数详情，如表 2-20 所示。







表 2-20

参数	类型	说明
jsonrpc	String	版本号
id	String	
method	String	方法名
params	Object[]	可变参数数组[地址, 密钥, ...]

提交的数据如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "method": "eth_getBalance",
  "params": ["0xa226084937afd728785a38347e4c24e8da59088d",
    "cd015e72e35848eab18a81bfff5081074", 60]
}
```

结果如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "result": true,
}
```

返回的结果信息如表 2-21 所示。

表 2-21

参数	类型	说明
jsonrpc	String	版本号
id	String	
result	String	解锁是否成功的标志位

## 4. 交易

请求路径：`http://ip+":"+port` POST

(1) 提交参数详情（主币），如表 2-22 所示。





表 2-22

参数	类型	说明
jsonrpc	String	版本号
id	String	
method	String	方法名
params	Object[]	可变参数数组[地址, ...]
gas	String	手续费数量上限
from	String	交易方地址
to	String	被交易方地址
value	String	交易金额
gasPrice	String	手续费价格（单价）
data	String	拼装数据（主币交易为空）

提交的数据如下：

```
{
  "method": "eth_sendTransaction",
  "id": "1",
  "jsonrpc": "2.0",
  "params": [{
    "gas": "0x15f90",
    "from": "0xa226084937afd728785a38347e4c24e8da59088d",
    "to": "0x897602a4C87dFBaea296fC86156A35058069EEB0",
    "value": "0x18de76816d8000",
    "gasPrice": "0x165a0bc00"
  }]
}
```

结果如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "result": "0xce06a7d9db9e583932fadde644ee3c90682c39fe7227068f2dd8972075632b95"
}
```

返回的结果信息如表 2-23 所示。





表 2-24

参数	类型	说明
jsonrpc	String	版本号
id	String	
method	String	方法名
params	Object[]	可变参数数组[地址, ...]
gas	String	手续费数量上限
from	String	交易方地址
to	String	代币合约地址
value	String	"0x0"固定值
gasPrice	String	手续费价格（单价）
data	String	拼装数据（methodId+转出地址+转出数量）

[illegible]



结果同上。

## 5. 查询交易信息

请求路径: `http://ip+":port POST`

(1) 提交参数详情 (根据 Hash 查询), 如表 2-25 所示。

表 2-25

参数	类型	说明
jsonrpc	String	版本号
id	String	
method	String	方法名
params	String[]	可变参数数组[Hash]

提交的数据如下:

```
{
  "method": "eth_getTransactionByHash",
  "id": "1",
  "jsonrpc": "2.0",
  "params": [
    "0xb903239f8543d04b5dc1ba6579132b143087c6
    8db1b2168786408fcbce568238"
  ]
}
```

结果如下:

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "hash": "0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab
    17f30a99d3cb96aed1d1055b",
    "nonce": "0x",
    "blockHash": "0xbeab0aa2411b7ab17f30a99d3cb9c6ef2fc5426
    d6ad6fd9e2a26a6aed1d1055b",
    "blockNumber": "0x15df",
    "transactionIndex": "0x1",
    "from": "0x407d73d8a49eeb85d32cf465507dd71d507100c1",

```





## 区块链：交易系统开发指南

```
    "to": "0x85h43d8a49eeb85d32cf465507dd71d507100c1",
    "value": "0x7f110",
    "gas": "0x7f110",
    "gasPrice": "0x09184e72a000",
    "input": "0x603880600c6000396000f300603880600c600
0396000f3603880600c6000396000f360"
  }
}
```

返回的结果信息如表 2-26 所示。

表 2-26

参数	类型	说明
jsonrpc	String	版本号
id	String	
nonce	String	发件人在此之前进行的交易次数
blockHash	String	当该交易处于 null 挂起状态时，其所在区块的散列值
blockNumber	String	当该交易处于 null 挂起状态时的区块号
transactionIndex	String	区块中交易指标位置，整数
from	String	发件人地址
to	String	接收人地址
value	String	交易数量
gas	String	发件人提供的手续费数量
input	String	数据与交易一起发送
gasPrice	String	手续费价格

(2) 提交参数详情（根据区块号和交易在区块中的位置查询），如表 2-27 所示。

表 2-27

参数	类型	说明
jsonrpc	String	版本号
id	String	
method	String	方法名
params	String[]	可变参数数组[blockNum,index(交易)]



提交的数据如下：

```
{
  "method": "eth_getTransactionByBlockNumberAndIndex",
  "id": "1",
  "jsonrpc": "2.0",
  "params": [ "0x29c", "0x0" ]
}
```

结果同上。

## 6. 查询最新区块高度

请求路径：http://ip+":port POST

提交参数详情，如表 2-28 所示。

表 2-28

参数	类型	说明
jsonrpc	String	版本号
id	String	
method	String	方法名
params	String[]	可变参数数组[]

提交的数据如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "method": "eth_blockNumber",
  "params": []
}
```

结果如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "result": "0x63500d26c"
}
```





返回的结果信息如表 2-29 所示。

表 2-29

参数	类型	说明
jsonrpc	String	版本号
id	String	
result	String	最新区块高度

## 7. 查询当前区块交易总数

请求路径：`http://ip+":"+port`

提交参数详情，如表 2-30 所示。

表 2-30

参数	类型	说明
jsonrpc	String	版本号
id	String	
method	String	方法名
params	String[]	可变参数数组[]

提交的数据如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "method": "eth_getBlockTransactionCountByNumber",
  "params": ["0x63500d26c"]
}
```

结果如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "result": "\"0x63"
}
```

返回的结果信息如表 2-31 所示。



表 2-31

参数	类型	说明
jsonrpc	String	版本号
id	String	
result	String	当前区块高度交易总数

## 8. 查询协议版本

请求路径：`http://ip+":"+port`

提交参数详情，如表 2-32 所示。

表 2-32

参数	类型	说明
jsonrpc	String	版本号
id	String	
method	String	方法名
params	String[]	可变参数数组[]

提交的数据如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "method": "eth_protocolVersion",
  "params": []
}
```

结果如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "result": "54"
}
```

返回的结果信息如表 2-33 所示。





表 2-33

参数	类型	说明
jsonrpc	String	版本号
id	String	
result	String	协议版本号

## 9. 查询预测gas值

请求路径：`http://ip+":"+port`

提交参数详情，如表 2-34 所示。

表 2-34

参数	类型	说明
jsonrpc	String	版本号
id	String	
method	String	方法名
params	String[]	可变参数数组[]

提交的数据如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "method": "eth_estimateGas",
  "params": []
}
```

结果如下：

```
{
  "jsonrpc": "2.0",
  "id": "1",
  "result": "\"0x5208\",//21000"
}
```

返回的结果信息如表 2-35 所示。



表 2-35

参数	类型	说明
jsonrpc	String	版本号
id	String	
result	String	评估 gas 值

## 2.3 SWT

### 2.3.1 SWT公有链的特点

在数字资产实现上，目前比特币系统无解，不能在它的上面构建数字资产；以太坊则走入了“唯智能合约论”，完全忽视智能资产无差别采取合约的 overhead。Ripple 采取银关 Fingate 模式，引入法币。

井通则是双向路线，采用银关模式管理简单的智能资产，在这里实现万物相连，万值相易，并且用类 OSPF 协议打造了高效的撮合算法。与此同时，井通也支持智能合约，对复杂的智能资产进行管理。采用这样的处理方式，是因为智能合约有 overhead，对于简单的智能资产是没必要浪费智能合约的功能的。

### 2.3.2 SWT公有链API接口

SWT 公有链的主要接口有创建钱包账户、查询账户余额、交易、查询交易信息、查询交易历史。下面通过 JSONRpc 请求方式进行讲解。

#### 1. 创建钱包账户

接口：/v2/wallet/new，GET 方法

例子：http://ip+":port/v2/wallet/new

结果如下：

```
{
  "success": true,
  "status_code": "0",
```



```

    "wallet": {
      "secret": "sshZGBQRZa2VDmtiPwGWU3aDYbDrQ",
      "address": "jhMWG7ah3R38PvhmtLaZnwNt4zgzyedAy9"
    }
  }
}

```

返回的结果信息如表 2-36 所示。

表 2-36

参数	类型	说明
success	Boolean	请求结果是否成功标志
wallet	Object	井通钱包
secret	String	井通钱包私钥
address	String	井通钱包地址

## 2. 查询账户余额

接口：/v2/accounts/{:address}/balances，GET 方法

必传的接口参数如表 2-37 所示。

表 2-37

参数	类型	说明
address	String	井通钱包地址

可选的接口参数如表 2-38 所示。

表 2-38

参数	类型	说明
currency	String	指定返回对应货币的余额，货币区分大小写
issuer	String	指定返回对应银关发行的货币

例子：http://ip+": "+port/v2/accounts/jKy8fMirqLzPU8XDML42wFG41Ci2LBrPkr/balances

结果如下：

```

{
  "success": true,
  "status_code": "0",
  "balances": [{

```





```

    "value": "29.97",
    "currency": "SWT",
    "issuer": "",
    "freezed": "30"
  },
  {
    "value": "1",
    "currency": "FDB",
    "issuer": "jGa9J9TkqtBcUoHe2zqhVFFbgUVED6o9or",
    "freezed": "0.000000"
  },
  {
    "value": "3",
    "currency": "DDB",
    "issuer": "jGa9J9TkqtBcUoHe2zqhVFFbgUVED6o9or",
    "freezed": "0.000000"
  }
],
"sequence": 4
}

```

返回的结果信息如表 2-39 所示。

表 2-39

参数	类型	说明
success	Boolean	请求结果
balances	Array	余额数组
value	String	余额
currency	String	货币名称, 3 个字母或 20 个字节的货币
issuer	String	货币发行方
freezed	String	冻结的金额
sequence	Integer	当前交易序列号 (用于本地签名)

### 3. 交易

接口: /v2/accounts/{:source\_address}/payments, POST 方法

接口参数如表 2-40 所示。



表 2-40

参数	类型	说明
source_address	String	交易方的井通地址

提交参数详情，如表 2-41 所示。

表 2-41

参数	类型	说明
secret	String	交易方的钱包私钥
client_id	String	此次请求的交易单号，交易单号需要唯一
payment	Object	交易对象
Source	String	发起账号
Destination	String	目标账号
Amount	Object	交易金额
choice	String	交易选择的 key，可选
memos	Array	交易备注，String 数组，可选

例子：http://ip+":"+port/v2/accounts/jQNdYXxgNHY49oxDL8mrjr7J6k7tdNy1kM/payments

提交的数据如下：

```
{
  "secret": "snxVJXMkURjrscL7gfwfWcywYzPkL",
  "client_id": "109",
  "payment": {
    "source": "jQNdYXxgNHY49oxDL8mrjr7J6k7tdNy1kM",
    "destination": "jD2RbZEpbG3T6iWDPyPiNBvpU8sAhRYbpZ",
    "amount": {
      "value": "1.00",
      "currency": "AAA",
      "issuer": "jMhLAPaNFo288PNo5HMC37kg6ULjJg8vPf"
    },
    "choice": "f53b09afcf9e1758a7b647f2f738c86426cabfc1",
    "memos": ["hello world", "hello payment"]
  }
}
```



结果如下：

```
{
  "success": true,
  "status_code": "0",
  "client_id": "109",
  "hash":
"0FE129880597A2681A7CDEA6098C19DA0EB97787FB09F0C1966AAD640D991BB5",
  "result": "tesSUCCESS",
  "fee": "0.000012"
}
```

返回的结果信息如表 2-42 所示。

表 2-42

参数	类型	说明
success	Boolean	请求结果
client_id	String	交易单号
hash	String	交易 Hash
result	String	交易的服务器结果，tesSUCCESS 表示成功，其他类型详见错误信息
fee	String	交易费用，并通计价

#### 4. 查询交易信息

接口：/v2/accounts/{:address}/payments/{:id}，GET 方法

接口参数如表 2-43 所示。

表 2-43

参数	类型	说明
address	String	交易用户的井通地址
id	String	交易的 Hash 或资源号

例子：`http://ip+":port/v2/accounts/jsqRs9BDCjyTuRWEpZk3yHa4MFmRi9D834/payments/D0BF25B015A6BF94F0645FBBC17B599E546B5DBECD5BBC23C1CB64F83C80A76B`

结果如下：

```
{
  "success": true,
```





## 区块链：交易系统开发指南

```
    "status_code": "0",
    "date": 1427828830,
    "hash":
"D0BF25B015A6BF94F0645FBBC17B599E546B5DBECD5BBC23C1CB64F83C80A76B",
    "type": "sent",
    "fee": "0.000012",
    "result": "tesSUCCESS",
    "memos": ["hello world", "hello payment"],
    "counterparty": "jsPVrt5C97gbNXx9S1DzBZvXETkWbmHgFQ",
    "amount": {
      "value": "1",
      "currency": "CNY",
      "issuer": "jsPVrt5C97gbNXx9S1DzBZvXETkWbmHgFQ"
    },
    "effects": []
  }
```

返回的结果信息如表 2-44 所示。

表 2-44

参数	类型	说明
success	Boolean	请求结果
date	Integer	交易时间，UNIXTIME 时间
hash	String	交易 Hash
type	String	交易类型，sent 或 received
fee	String	交易费用
result	String	交易的服务器结果
memos	Array	交易备注，String 数组
counterparty	String	交易对家
amount	Object	交易金额
effects	Array	交易效果

## 5. 查询交易历史

接口：/v2/accounts/{:address}/payments，GET 方法

必传的接口参数如表 2-45 所示。



表 2-45

参数	类型	说明
address	String	交易用户的井通地址

可选的接口参数如表 2-46 所示。

表 2-46

参数	类型	说明
results_per_page	Integer	返回的每页数据量，默认每页 10 项
page	Integer	页码，默认从第一页开始
marker	Object	交易记录标记，表示从当前记录继续向下查找（注：marker 的优先级大于 page。即当有 marker 和 page 时，表示从 marker 处向下查找找到第 page 页）
ledger	Integer	账本号
seq	Integer	上一次查询返回的最后一个交易号

例子：`http://ip+":port/v2/accounts/jnbsmZpkBaGGpPip2A3HujzzWcQvURNGC4/payments?results_per_page=1&page=4`

结果如下：

```
{
  "success": true,
  "status_code": "0",
  "marker": {
    "ledger": 7657903,
    "seq": 0
  },
  "payments": [{
    "date": 1430271890,
    "hash": "9600FFB5966458EB32D6C5344D3F720A59A210AA6ECF92F68D2D7C077C242375",
    "type": "sent",
    "fee": "0.000012",
    "result": "tesSUCCESS",
    "memos": ["hello world", "hello payment"],
    "counterparty": "jP659KPCEa76rf7kUCwoTk9m5ZjGiRXco7",
    "amount": {
      "value": "0.01",
```



```

        "currency": "8000000001201504081454405641668733216531",
        "issuer": "jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS"
    },
    "effects": []
  }
}

```

返回的结果是下一页的数据，返回的结果信息如表 2-47 所示。

表 2-47

参数	类型	说明
success	Boolean	请求结果
marker	Object	交易记录标记，是当前交易记录的结束标记，也是下一次查找的开始标记；当 marker 存在时，表示后续还有交易记录
hash	String	交易 Hash

## 2.4 MOAC

### 2.4.1 MOAC公有链的特点

MOAC 公有链的创新之处在于智能合约的异步调用和分片处理，该项目旨在提供一种可扩展且有弹性的区块链，支持基于分层结构的状态交易、数据访问和控制流程。它创建了一个框架以允许用户采用高效的方式执行智能合约。它还提供了开放的体系结构，采用底层基础设施来快速、简便地产生子区块链。

### 2.4.2 MOAC公有链API接口

MOAC 公有链的主要接口有查询账户余额、查询当前区块高度、查询区块详情、交易、查询交易详情、查询区块交易总数、查询交易收据。下面通过 JavaScript 请求方式进行讲解。

#### 1. 查询账户余额

例子：

```
var balance = chain3.mc.getBalance("0x407d73d8a49eeb8
```





```
5d32cf465507dd71d507100c1");
console.log(balance); // instanceof BigNumber
console.log(balance.toString(10)); // '1000000000000'
console.log(balance.toNumber()); // 1000000000000
```

## 2. 查询当前区块高度

例子:

```
var number = chain3.mc.blockNumber ;
console.log(number); // 2744
```

## 3. 查询区块详情

例子:

```
var info = chain3.mc.getBlock(0);
console.log(info);
```

结果如下:

```
{
  "extraData": "0x31393639415250414e4554373354435049
503039425443323031384d4f4143",
  "gasLimit": 9000000,
  "gasUsed": 0,
  "hash": "0x6b9661646439fab926ffc9bccdf3abb572d5
209ae59e3390abf76aee4e2d49cd",
  "logsBloom": "0x00000000000000000000000000000000
000000000000000000000000",
  "miner": "0x000000000000000000000000000000000000",
  "mixHash": "0x00000000000000000000000000000000",
  "nonce": "0x0000000000000042",
  "number": 0,
  "parentHash": "0x00000000000000000000000000000000
000000000000000000000000",
  "receiptsRoot": "0x56e81f171bcc55a6ff8345e692c0f86e5b
48e01b996cad001622fb5e363b421",
  "sha3Uncles": "0x1dcc4de8dec75d7aab85b567b6ccd41
d312451b948a7413f0a142fd40d49347",
  "size": 540,
```



```

    "stateRoot": "0xe221c9e4ad19514d7ce3e6b0bec3ad7f6cc
293336e59c301cda293cfbda83df6",
    "timestamp": 0,
    "transactions": [],
    "transactionsRoot": "0x56e81f171bcc55a6ff8345e692c0f86e5b4
8e01b996cad001622fb5e363b421",
    "uncles": []
}

```

返回的结果信息如表 2-48 所示。

表 2-48

参数	类型	说明
hash	String	区块的散列
number	Number	区块号
gasLimit	Number	区块允许的最高手续费
gasUsed	Number	区块在交易中使用的费用
nonce	String	生成的工作量证明的散列值
size	BigNumber	以字节为单位整数区块的大小
extraData	String	区块的额外数据字段
Miner	String	获得采矿奖励的受益人的地址

## 4. 交易

例子：

```

var code = "603d80600c6000396000f3007c01000000000000
0000000000000000000000000000000000000000000000000
00006000350463c6888fa18114602d57005b60076
00435028060005260206000 f3 ";

chain3.mc.sendTransaction({
  data: code
}, function(err, address) {
  if (!err)
    console.log(address);
});

```



提交参数详情，如表 2-49 所示。

表 2-49

参数	类型	说明
code	Object	参数对象
from	String	支付账户地址
to	String	目标地址
value	String	交易金额
gas	String	用于交易的手续费
gasPrice	String	交易的手续费
data	String	包含消息关联数据的字节字符串
nonce	Number	一个随机的整数

返回的结果信息如表 2-50 所示。

表 2-50

参数	类型	说明
address	String	交易的散列

## 5. 查询交易详情

(1) 根据交易 Hash 查询。

例子：

```
var txhash = "0x687751dd47684f4b5df263ae4ec39f54f057d0e2a1dde56f9d52766849c9c7fe";
var transaction = chain3.mc.getTransaction(txhash);
console.log(transaction);
```

结果如下：

```
{
  blockHash: '0x716c602d49055b4e24fbe7da952c1ad81820a
d0401f3cb3ce12e832fbcc368f5',
  blockNumber: 57259,
  from: '0x7cfd775c7a97aa632846eff35dcf9dbcf502d0f3',
  gas: 1000,
  hash: '0xf1c1771204431c1c584e793b49d41586a923
```





```
c370be93673aac42d66252bc8d0a',
  input: '0x',
  nonce: 840,
  syscnt: '0x0',
  to: '0x3435410589ebd06b74079a1e141759d8502aeb8b',
  transactionIndex: 689,
  shardingFlag: '0x0'
}
```

返回的结果信息如表 2-51 所示。

表 2-51

参数	类型	说明
hash	String	交易的散列
nonce	Number	发件人在此之前进行的交易次数
transactionIndex	Number	区块中交易指标位置，整数
input	String	与交易一起发送的数据

(2) 根据交易区块号和交易在区块中的位置查询。

例子：

```
var transaction = chain3.mc.getTransactionFromBlock(921, 2);
console.log(transaction); // see chain3.mc.getTransaction
```

结果同上。

## 6. 查询区块交易总数

例子：

```
var number = chain3.mc.getBlockTransactionCount("0xddf
b8508bff841242099e640efe59f5e5252bela60fa701d333e1a8bfdee6263");
console.log(number); // 1
```

参数为区块的散列。

## 7. 查询交易收据

例子：

```
var receipt = chain3.mc.getTransactionReceipt('
```



```
0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b');  
console.log(receipt);
```

结果如下：

```
{  
  "transactionHash": "0x9fc76417374aa880d4449a1f7f31ec597f00b1f  
6f3dd2d66f4c9c6c445836d8b",  
  "transactionIndex": 0,  
  "blockHash": "0xef95f2f1ed3ca60b048b4bf67cde2195961e0  
bba6f70bcbea9a2c4e133e34b46",  
  "blockNumber": 3,  
  "contractAddress": "0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b",  
  "cumulativeGasUsed": 314159,  
  "gasUsed": 30234  
}
```

返回的结果信息如表 2-52 所示。

表 2-52

参数	类型	说明
transactionHash	String	交易的散列
blockNumber	Number	交易所在的区块号
transactionIndex	Number	区块中交易指标位置，整数
contractAddress	String	创建合同地址
cumulativeGasUsed	Number	在区块中执行交易时使用的手续费

## 2.5 EOS

### 2.5.1 EOS公有链的特点

EOS 作为叫板以太坊的公有链，最有技术特点的地方在于可以简化用户账号的生成与管理，并且能恢复账号，这在用户看来提高了安全性，并且号称支持百万级 TPS 的交易速度也让其赚足了公众的眼球，完全零费率，并可以快速且容易地部署去中心化应用。



## 2.5.2 EOS公有链API接口

EOS 公有链的主要接口有创建钱包账户、查询账户信息、解锁账户、交易。下面通过 JSONRpc 请求方式进行讲解。

### 1. 创建钱包账户

接口：`http:// ip + ":" + port/v1/wallet/create`

提交参数详情，如表 2-53 所示。

表 2-53

参数	类型	说明
params	String	账户名

提交的数据如下：

```
{
  "params": "weiqingwei"
}
```

结果如下：

```
{
  "result": "PW5KFWYKqvt63d4iNvedfDEPVZL227D3
RQ1zpVFzuUwhMAJmRAYyX",
}
```

返回的结果信息如表 2-54 所示。

表 2-54

参数	类型	说明
result	String	返回账户名对应的密钥

### 2. 查询账户信息

接口：`http:// ip + ":" + port/v1/chain/get_account`

提交参数详情，如表 2-55 所示。





表 2-55

参数	类型	说明
account_name	String	账户名

提交的数据如下：

```
{
  "account_name": "weiqingwei",
}
```

结果如下：

```
{
  "name": "inita",
  "eos_balance": "999998.9574 EOS",
  "staked_balance": "0.0000 EOS",
  "unstaking_balance": "0.0000 EOS",
  "last_unstaking_time": "2106-02-07T06:28:15",
  "permissions": [{
    "name": "active",
    "parent": "owner",
    "required_auth": {
      "threshold": 1,
      "keys": [{
        "key": "EOS6MRyAjQq8ud7hVNYcfnVPJqcVpscN5So8B
htHuGYqET5GDW5CV",
        "weight": 1
      }],
      "accounts": []
    }
  }],
  {
    "name": "owner",
    "parent": "owner",
    "required_auth": {
      "threshold": 1,
      "keys": [{
        "key": "EOS6MRyAjQq8ud7hVNYcfnVPJqcVpscN5So8Bh
tHuGYqET5GDW5CV",
        "weight": 1
      }],

```



```

        "accounts": []
    }
}
]
}

```

返回的结果信息如表 2-56 所示。

表 2-56

参数	类型	说明
name	String	账户名
eos_balance	String	账户余额

### 3. 解锁账户

接口：http://ip + ":" + port/v1/wallet/unlock

提交参数详情，如表 2-57 所示。

表 2-57

参数	类型	说明
params	String[]	条件参数[账户名, 密钥]

提交的数据如下：

```

{
    "params": "[\"weiqingwei\", \"PW5KFWYKqvt63d4iNvedfDEPVZL227D3RQ1zpVFzuUwhMAJmRAYyX\"]"
}

```

结果如下：

```

{
    "result": true
}

```

返回的结果信息如表 2-58 所示。

表 2-58

参数	类型	说明
result	Boolean	解锁是否成功的标志位



## 4. 交易

接口：`http://ip + ":" + port/v1/chain/push_transaction`

提交参数详情，如表 2-59 所示。

表 2-59

参数	类型	说明
params	Object[]	条件参数[交易用户名, 被交易用户名, 交易金额]

提交的数据如下：

```
{
  "params": ["inita", "initb", 9]
}
```

结果如下：

```
{
  "transaction_id": "5294319392180bb27ef99ff7454abd096de
c02c791266261ebc73db2eed2cbea",
  "processed": {
    "refBlockNum": "1233",
    "refBlockPrefix": "2923796072",
    "expiration": "2017-07-30T11:53:37",
    "scope": [
      "inita",
      "initb"
    ],
    "signatures": [],
    "authorizations": [{
      "account": "inita",
      "permission": "active"
    }]
  },
  "messages": [{
    "code": "eos",
    "type": "transfer",
    "data": {
      "from": "inita",
      "to": "initb",
      "amount": 9
    }
  ]
}
```





```
    },  
    "hex_data": "000000008040934b0000000000041934b  
090000000000000000"  
  }  
}  
}
```

返回的结果信息如表 2-60 所示。

表 2-60

参数	类型	说明
transaction_id	Boolean	解锁是否成功的标志位
from	String	交易方
to	String	收款方
amount	Long	交易金额

## 2.6 本章小结

本章主要讲解了 BTC、ETH、SWT、MOAC、EOS 公有链的特点和 API 调用，读者通过阅读本章内容对这些公有链会有一个基本的认识。



# 3

## 第 3 章

---

### 交易系统架构

#### 3.1 系统概述

---

##### 3.1.1 背景

---

由于市场性质不同，数字货币交易平台和传统的金融交易平台在业务营运方式上有较大的差异。数字货币是一种数字资产，它的价格行情会受到全球市场的影响，启动了金融杠杆交易后这种价格波动影响被放大了多倍，这使得交易系统必须满足全年  $7 \times 24$  小时交易服务不间断，因为系统服务的中断意味着行情将产生剧烈的波动，加大交易的风险。因此，在系统设计上需要尽可能避免停机维护，从硬件到软件实现完整的冗余和高可用性，每一个系统都充分围绕着高可用性和容错来设计和实现，必须要做到所有系统模块的高可用性和核心交易系统的快速故障转移。



中心化交易所的交易模式可以分为以成交单位为主导的场内交易所和以报价单为主导的场外交易所。因为进行场外交易双方交易不透明，交易存在诸多风险和不可控因素，所以人们一般选择场内交易所，只有在特殊情况下才进行场外交易。中心化的场内交易需要具有用户账号管理、KYC、资产充值、资产托管、交易撮合、资产清算、资产提现、履约担保、上架新币等功能，然而在每一个环节都有可能存在风险和问题，从而给用户带来安全隐患。自从 MTGOX 比特币失窃以来，交易所的安全问题频发，归根到底，是由于不可信的中心化交易所作为第三方背书，不可避免地出现了内部人运营、黑客攻击、资产挪用、上架新币不受监督等诸多安全风险。

相对于中心化交易所，去中心化交易所的业务要简单得多，用户资产时刻处于用户自己或开源的智能合约代码控制之下，用户不需要向交易所提供个人信息，资产转移不需要任何人工审核。某些去中心化交易所本身是去中心化的，但它在资金和代币充提处设置了中心化的承兑商，承兑商通过收取一定的手续费提供资金和代币的承兑服务，从而导致这些交易所陷入了人气低迷、交易深度不够，并且很多代币无法购买的窘境。

### 3.1.2 系统目标

交易系统的目标是对平台的核心数据访问进行封装，为前面的各个业务网元提供访问核心数据的业务方法，让各个网元无须关心数据库的物理模型、存储及数据库事务管理，只专注自己的业务逻辑实现。

所有的系统架构都为高可用性做了大量的设计，在前端 Web 层面、后台数据缓存和业务服务层面均允许任意节点失效。在数据库层面通过复制和数据分区的方式实现了主备层面的高可用性，在出现故障后通过相应的业务日志检查，即可迅速通过 IP 漂移实现数据库的故障恢复。

### 3.1.3 设计理念

#### 1. 以空间换时间

##### (1) 多级缓存，静态化

- ◎ 客户端页面缓存（HTTP header 中包含 Expires/Cache of Control、last modified（304，Server 不返回 body，客户端可以继续用 Cache，减少流量）、ETag）。
- ◎ 反向代理缓存。





- ◎ 应用端的缓存（Memcache）。
- ◎ 内存数据库。
- ◎ Buffer、Cache 机制（数据库、中间件等）。

## （2）索引

- ◎ 哈希索引适合于综合数组的寻址和链表的插入特性，可以实现数据的快速存取。
- ◎ B 树索引适合于以查询为主导的场景，避免多次 I/O，可以提高查询效率。
- ◎ 倒排索引实现单词到文档映射关系的最佳实现方式和最有效的索引结构，被广泛用在搜索领域。
- ◎ Bitmap 是一种非常简捷、快速的数据结构，它能同时使存储空间和速度最优化（而不必以空间换时间），适合于海量数据的计算场景。

## 2. 并行与分布式计算

### （1）任务切分，分而治之（MR）

在大规模的数据中，数据存在一定的局部性特征，利用局部性原理将海量数据计算的问题分而治之。

MR 模型是无共享的架构，数据集分布至各个节点。在处理时，每个节点就近读取本地存储的数据进行处理（Map），并将处理后的数据进行合并（Combine）、排序（Shuffle and Sort）后再分发（至 Reduce 节点），避免了大量数据的传输，提高了处理效率。

### （2）多进程、多线程并行执行（MPP）

并行计算（Parallel Computing）是指同时使用多种计算资源解决计算问题的过程，是提高计算机系统计算速度和处理能力的一种有效手段。它的基本思想是用多个处理器/进程/线程来协同求解同一个问题，即将被求解的问题分解成若干部分，各部分均由一个独立的处理器来并行计算。

MPP 和 MR 的区别在于，它是基于问题分解的，而不是基于数据分解的。

## 3. 多维度可用

### （1）负载均衡、容灾、备份

随着平台并发量的增大，需要扩容集群节点，利用负载均衡设备进行请求的分发。通



常负载均衡设备在提供负载均衡的同时，也提供了失效检测功能。为了提高可用性，需要有容灾备份，以防止节点宕机失效带来不可用问题。备份分为在线和离线两种，可以根据失效性要求的不同，选择不同的备份策略。

## （2）读写分离

读写分离是针对数据库来讲的，随着系统并发量的增大，提高数据访问可用性的一个重要手段就是写数据和读数据分离。当然，在读写分离的同时，需要关注数据的一致性问题；对于一致性问题，在分布式系统 CAP 定理中应更多地关注可用性。

## （3）依赖关系

平台中各个模块之间的关系尽量低耦合，可以通过相关的消息组件进行交互，能异步的则异步，分清楚数据流转的主流程和副流程，主、副是异步的。比如记录日志可以异步操作，以增加整个系统的可用性。

当然，在异步处理中，为了确保数据被接收或处理，往往需要确认机制（Confirm、Ack）。然而，在有些场景中，虽然请求已经得到处理，但是因其他原因（比如网络不稳定）确认消息没有返回，那么在这种情况下就需要进行请求的重发，对请求的处理设计因重发因素需要考虑幂等性。

## （4）监控

监控也是提高整个平台可用性的一个重要手段，多平台进行多个维度的监控；模块在运行时是透明的，以达到运行期白盒化。

# 4. 伸缩性

## （1）拆分

拆分包括对业务的拆分和对数据库的拆分。

系统的资源是有限的，对于执行时间比较长的业务，如果采用一竿子执行的方式，在大量并发操作下，这种阻塞方式无法有效地及时释放资源给其他进程使用，从而导致系统的吞吐量不高。因此需要对业务进行逻辑分段，采用异步非阻塞方式，提高系统的吞吐量。

随着数据量和并发量的增大，读写分离不能满足系统并发性能的要求，需要对数据进行切分，包括对数据进行分库和分表。这种分库分表的方式，需要增加对数据的路由逻辑支持。



## （2）无状态

对于系统的伸缩性而言，模块最好是无状态的，通过增加节点就可以提高整个系统的吞吐量。

## 5. 优化资源的利用

### （1）系统容量有限

系统的容量是有限的，其承受的并发量也是有限的，所以在进行架构设计时，一定要考虑流量控制，防止因意外攻击或者瞬时并发量的冲击导致系统崩溃。比如可以考虑对请求进行排队，如果超出预期的范围，则可以进行告警或者丢弃。

### （2）原子操作和并发控制

对于对共享资源的访问，为了防止冲突，需要进行并发控制，同时对于有些交易需要通过事务性来保证一致性。所以在进行交易系统设计时，需要考虑原子操作和并发控制。

保证并发控制常用的一些高性能手段有乐观锁、Latch、Mutex、写时复制、CAS等；多版本并发控制（MVCC）通常是保证一致性的重要手段，它在数据库设计中经常被用到。

### （3）基于不同的逻辑，采取不同的策略

平台中的业务逻辑存在不同的类型，有计算复杂型的、有消耗 I/O 型的，并且就同一种类型而言，不同的业务逻辑消耗的资源数量也是不一样的，这就需要针对不同的逻辑采取不同的策略。

针对消耗 I/O 型的业务逻辑，可以采取基于事件驱动的异步非阻塞方式，单线程方式可以减少线程切换引起的开销，或者在多线程的情况下采取自旋（Spin）的方式，减少对线程的切换（比如 Oracle Latch 设计）；对于计算复杂型的业务逻辑，可以充分利用多线程进行操作。

对于同一种类型的调用方式，对不同的业务进行合适的资源分配，设置不同的计算节点数量或者线程数量，对业务进行分流，优先执行优先级高的业务。

### （4）容错隔离

当系统的有些业务模块出现错误时，为了减少在并发情况下对正常请求处理的影响，有时候需要考虑对这些处于异常状态的请求进行单独渠道的处理，甚至暂时自动禁止这些异常的业务模块。

有些请求的失败可能是偶然的、暂时的（比如网络不稳定导致的失败），需要考虑进





行请求重试。

### （5）资源释放

系统的资源是有限的，使用完后一定要在最后释放资源，而不管请求走的是正常路径还是异常路径，以便于资源的及时回收，供其他请求使用。

### （6）超时

在设计通信架构时，往往需要考虑超时控制。

在接口调用过程中，Consumer 调用 Provider，Provider 在响应时有可能会慢，如果 Provider 10s 响应，那么 Consumer 也会至少 10s 才响应。如果出现这种情况的频度很高，那么就会整体降低 Consumer 端服务的性能。

这种响应时间慢的症状就会像一层一层波浪一样，从底层系统一直涌到最上层，造成整个链路的超时。所以，Consumer 不可能无限制地等待 Provider 接口的返回，它会设置一个时间阈值，如果超过了该阈值，就不继续等待了。

对超时时间的选取，一般看 Provider 正常响应时间是多少，再追加一个 Buffer 即可。

### （7）重试

设置超时时间是为了保护服务，避免因为 Provider 响应慢，Consumer 服务也变得响应很慢，这样 Consumer 就可以尽量保持原有的性能。

但是也有可能 Provider 只是偶尔抖动，如果超时后直接放弃，不做后续处理，就会导致当前请求错误，也会带来业务方面的损失。所以对于这种偶尔抖动，可以在超时后进行请求重试，如果重试时正常返回了，那么这次请求就被挽救了，能够正常给前端返回数据，只不过比原来响应慢一点。

重试时的细化策略是：重试时可以考虑换一台机器来进行调用，因为原来的机器可能由于临时负载高而性能下降，重试会加剧其性能问题，而换一台机器更快返回的概率也更大一些。

### （8）幂等

如果允许 Consumer 重试，那么 Provider 就要能够做到幂等。即同一个请求被 Consumer 多次调用，对 Provider 产生的影响（一般指与某些写入相关的操作）是一致的。而且这个幂等应该是服务级别的，而不是某台机器层面的，重试调用任何一台机器都应该做到幂等。



### （9）熔断

重试是为了应付偶尔抖动的情况，以求更多地挽回损失。可是如果 Provider 持续的响应时间超长，该怎么办呢？

如果 Provider 是核心路径的服务，宕掉基本就没法提供服务了，那我们也没办法。但如果是一个不那么重要的服务，却因为这个服务响应时间长导致 Consumer 里面的核心服务也被拖慢，那就得不偿失了。

一般超时时间都比平均响应时间长一些，现在所有到达 Provider 的请求都超时了，那么 Consumer 请求 Provider 的平均响应时间就等于超时了，系统变慢，服务的响应能力下降。

而重试则会加重这种问题，使 Consumer 的可用性变得更差。因此就出现了熔断的逻辑，即如果检查出来频繁超时，就把 Consumer 调用 Provider 的请求直接短路掉，不实际调用，而是直接返回一个 mock 的值。等 Provider 服务恢复稳定之后，再重新调用。

### （10）限流

上面几种策略都是 Consumer 针对 Provider 出现的各种情况而设计的。而 Provider 有时候也要防范来自 Consumer 的流量突变问题。

比如有这样一个场景：Provider 是一个核心服务，给  $N$  个 Consumer 提供服务，突然某个 Consumer 的流量飙升，占用了 Provider 大部分机器时间，导致其他可能更重要的 Consumer 不能被正常服务。

对于这样的场景，Provider 端需要根据 Consumer 的重要程度，以及平时的 QPS 大小，给每个 Consumer 设置一个流量上限，在同一时间内只会给一个 Consumer 提供  $N$  个线程支持，超过限制则等待或者直接拒绝。

### （11）资源隔离

Provider 可以对从 Consumer 来的流量进行限制，防止 Provider 被拖垮。同样，Consumer 也需要对调用 Provider 的线程资源进行隔离，这样可以确保调用某个 Provider 逻辑不会耗光整个 Consumer 的线程池资源。

### （12）服务降级

降级服务既可以通过代码自动判断，也可以根据突发情况人工切换。



## 3.2 业务功能

交易系统业务功能示意图如图 3-1 所示。

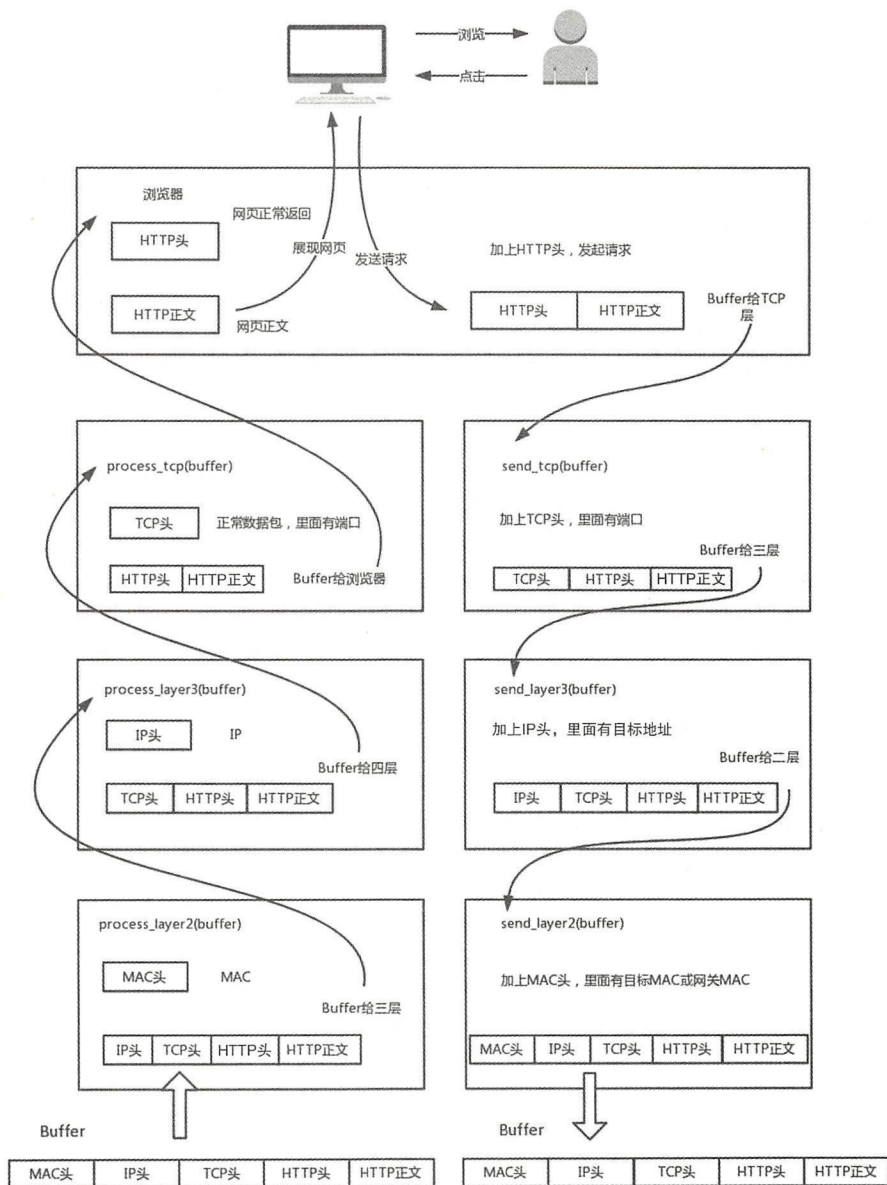


图 3-1



### 3.2.1 功能架构

交易系统功能架构示意图如图 3-2 所示。

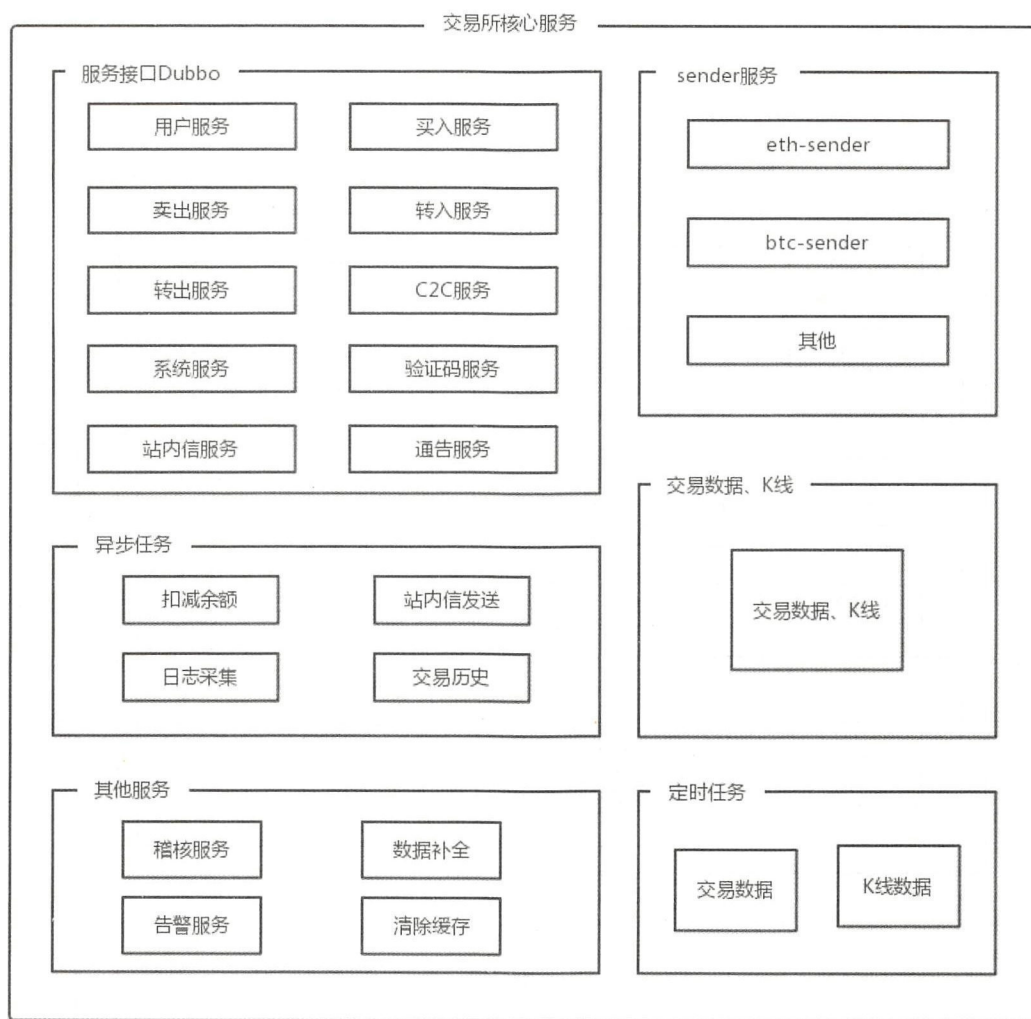


图 3-2



## 3.2.2 功能模块

交易系统功能模块示意图如图 3-3 所示。

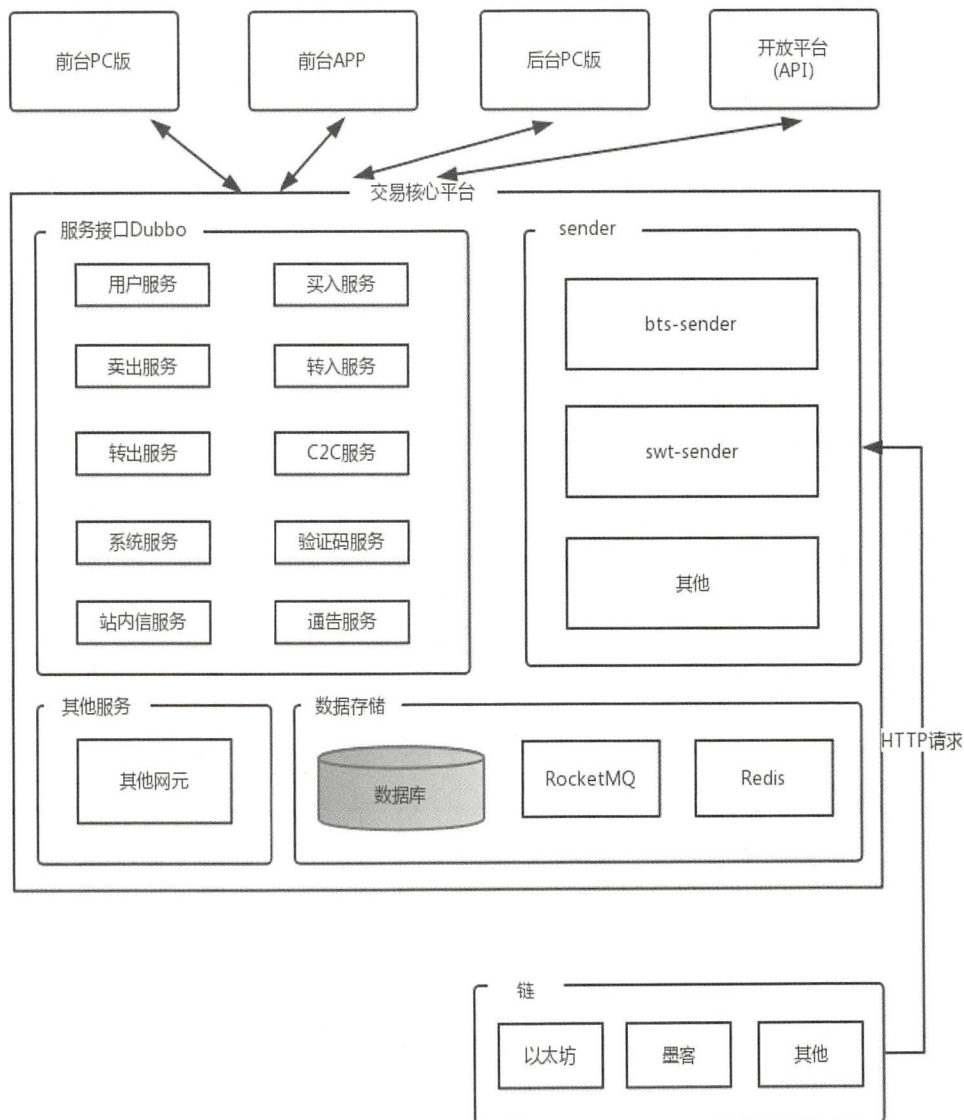


图 3-3



### 3.2.3 系统流程图

交易系统流程图如图 3-4 所示。

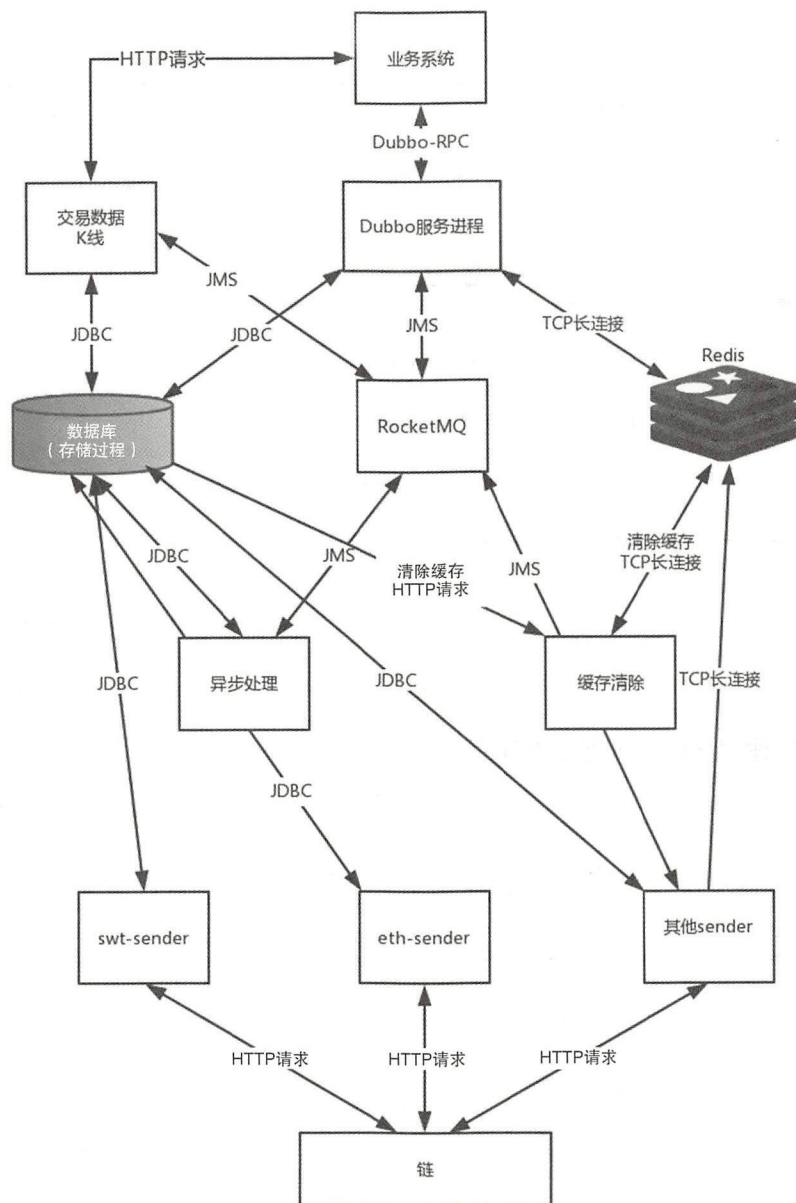


图 3-4





## 3.2.4 业务流程

### 1. 服务调用时序图（如图 3-5 所示）

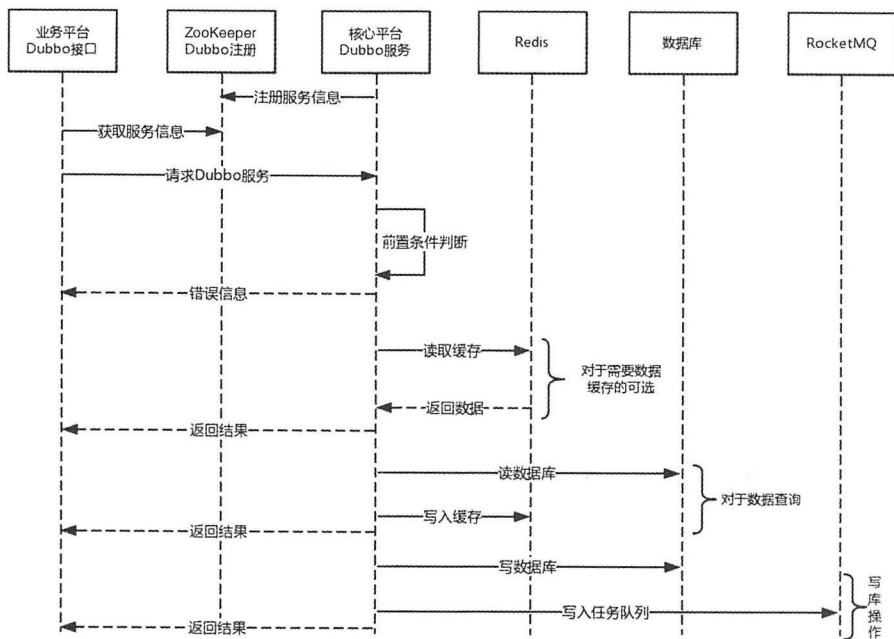


图 3-5

### 2. 异步处理时序图（如图 3-6 所示）

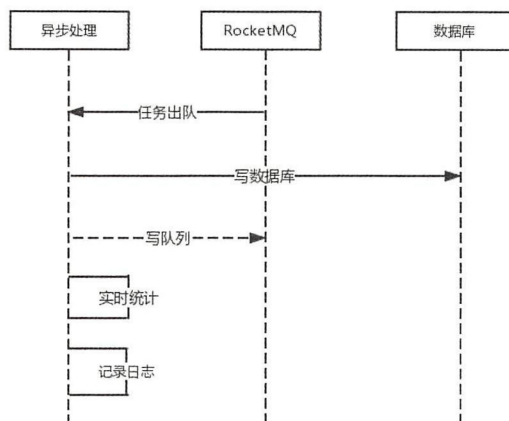


图 3-6



### 3. K线、交易数据时序图（如图 3-7 所示）

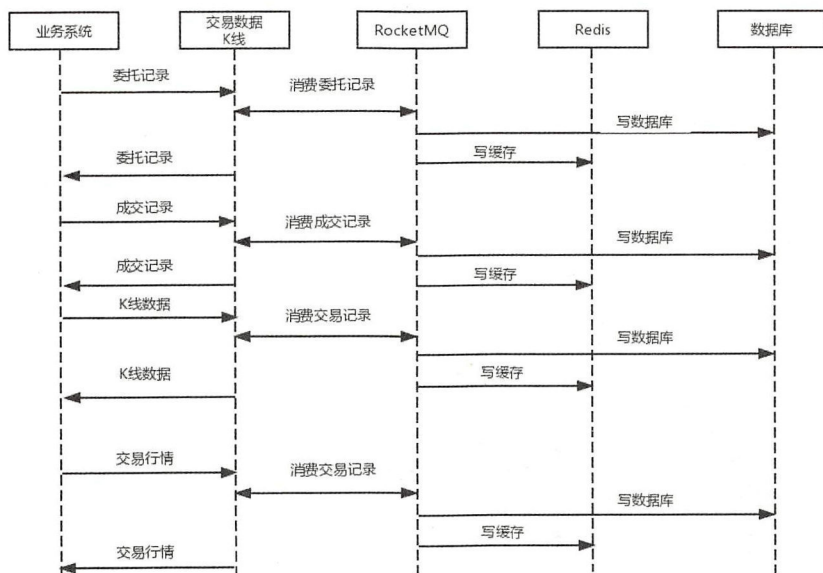


图 3-7

### 4. 用户交易时序图（如图 3-8 所示）

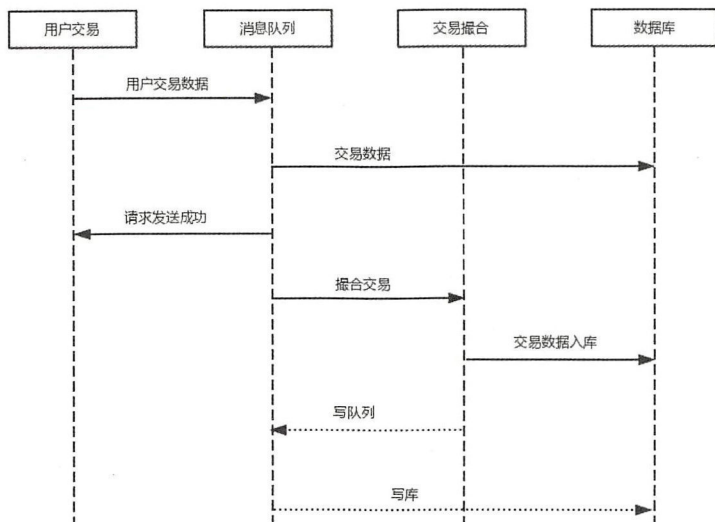


图 3-8



## 5. 服务端架构推荐（如图 3-9 所示）

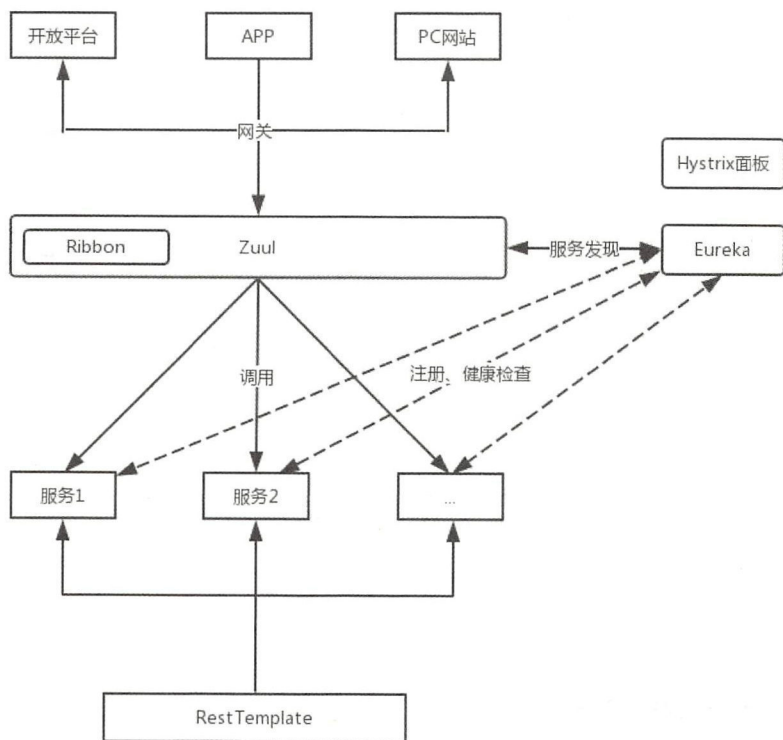


图 3-9

- ⊙ 服务在 Eureka 实例中注册，由 Spring 管理 Bean 来发现和调用。
- ⊙ 通过配置方式可以启动嵌入式的 Eureka 服务器。
- ⊙ Feign 客户端通过声明方式即可导入服务代理。
- ⊙ Zuul 使用 Ribbon 服务实现客户端的负载均衡。
- ⊙ 通过声明方式即可插入 Hystrix 面板服务器。
- ⊙ 在 Spring 环境中可以直接配置 Netflix 的组件。
- ⊙ Zuul 可以自动注册过滤器和路由器，形成一个反向代理服务器。
- ⊙ Hystrix 面板可以对服务的状态进行监控，并提供容错机制。





## 3.3 系统模块

### 3.3.1 服务熔断

当服务的输入负载迅速增加时,如果没有采取有效的措施对负载进行熔断,则会使服务迅速被压垮,进而导致所依赖的服务也会被压垮,出现雪崩效应,因此需要在服务架构中实现熔断模式(如图3-10所示)。

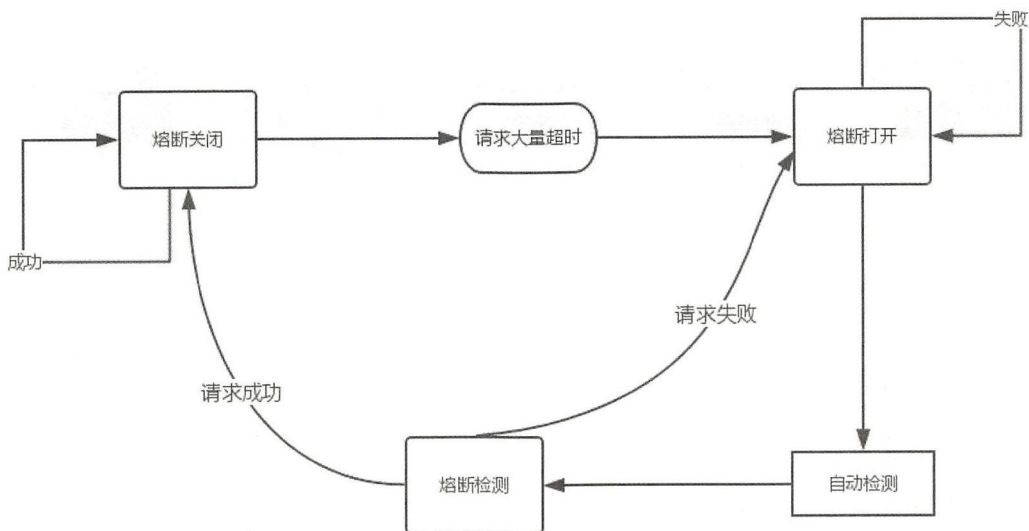


图 3-10

### 3.3.2 风控服务

由于不能像传统的证券交易所那样当天卖出证券资产后第二天才能提现,因为基于区块链的数字货币领域太新,在没有法律认可的第三方资金监管的情况下,用户希望自己的资产随时掌握在手里才踏实,这就带来了用户对法币和虚拟货币充值提现的实时结算需求,使结算和风控系统的设计也变得非常复杂,实现结算提现加数字资产的不可追回特性,导致出现任何问题都是非常致命的,因此相对传统金融平台来说其在技术上存在更大的挑战。



### 3.3.3 数据库设计

整个系统主要分为 5 个库，如图 3-11 所示。

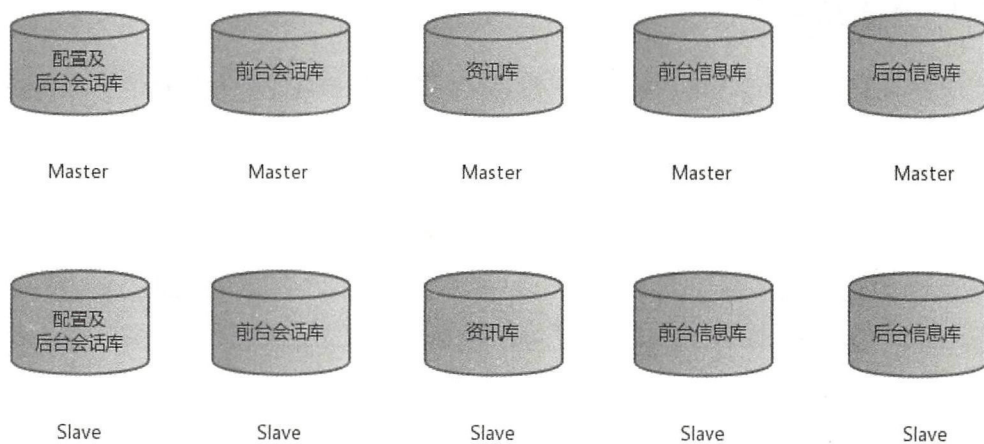


图 3-11

- ① 配置及后台会话库：主要将系统的基础配置，以及后台用户的会话信息表等数据记录在该库中。
- ② 前台会话库：存储前台用户、会话等信息。
- ③ 资讯库：存储公告、资讯等信息。
- ④ 前台信息库：前台用户库表。
- ⑤ 后台信息库：后台用户库表。

### 3.3.4 组网部署结构设计

系统组网部署结构设计图如图 3-12 所示。



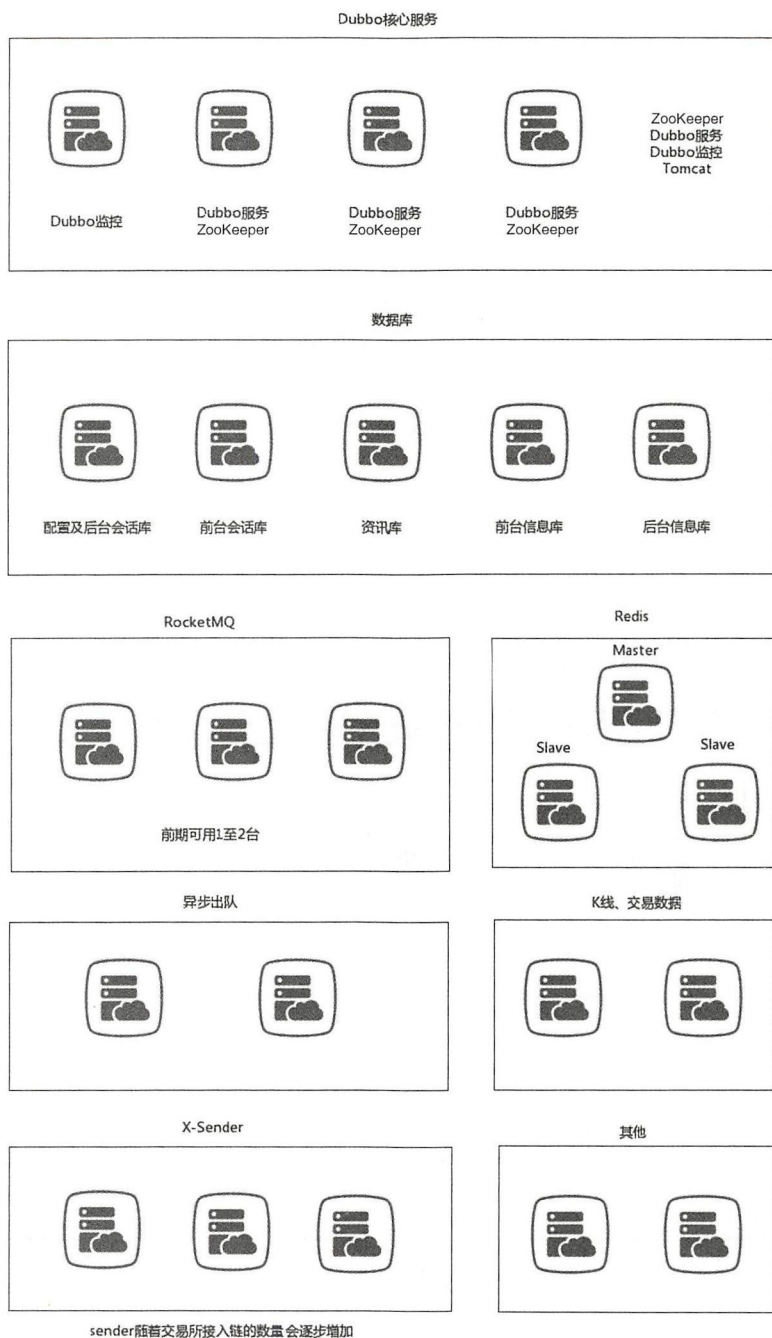


图 3-12





## 3.4 技术选型

### 3.4.1 ZooKeeper选型

#### 1. ZooKeeper原理

ZooKeeper 是一个开源的分布式协调服务，它为分布式应用提供了高效且可靠的分布式协调服务，以及统一命名空间服务、配置服务和分布式锁等分布式基础服务。

集群角色如下：

- ① ZooKeeper 中包含 Leader、Follower 和 Observer 三个角色。
- ② 通过一次选举过程，被选举的机器节点称为 Leader，Leader 机器为客户端提供读写服务。
- ③ Follower 和 Observer 是集群中的其他机器节点，它们唯一的区别就是 Observer 不参与 Leader 的选举过程，也不参与写操作的过半写成功策略。

一个典型的 ZooKeeper 集群示意图如图 3-13 所示。

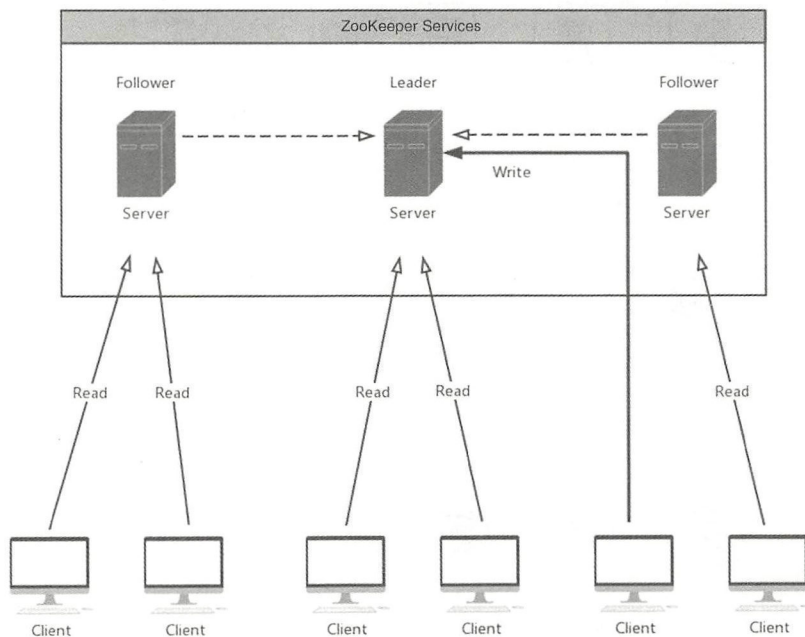


图 3-13



## 2. 设计目的

- ◎ 最终一致性：不论客户端（Client）连接到哪台服务器（Server），展示给它的都是同一个视图。这是 ZooKeeper 最重要的性能。
- ◎ 可靠性：具有简单、健壮、良好的性能，如果一条消息被一台服务器接收，那么它将被所有的服务器接收。
- ◎ 实时性：ZooKeeper 保证客户端在一个时间间隔范围内获得服务器的更新信息，或者服务器的失效信息。但由于网络延时等原因，ZooKeeper 不能保证两个客户端同时得到刚更新的数据，如果需要最新数据，则应该在读数据之前调用 sync()接口。
- ◎ 等待无关（wait-free）：慢的或者失效的客户端不得干预快速的客户端的请求，使得每个客户端都能有效等待。
- ◎ 原子性：更新只能成功或者失败，没有中间状态。
- ◎ 顺序性：包括全局有序和偏序两种。全局有序是指在一台服务器上如果消息 a 在消息 b 前被发布，那么在所有服务器上消息 a 都将在消息 b 前被发布；偏序是指如果消息 b 在消息 a 后被同一个发送者发布，那么消息 a 必将排在消息 b 的前面。

## 3. 选主流程

当 Leader 崩溃或者 Leader 失去大多数的 Follower 时，ZooKeeper 进入恢复模式，这时需要重新选举出一个新的 Leader，让所有的服务器都恢复到一个正确的状态。ZooKeeper 的选举算法有两种：一种是基于 Basic Paxos 算法实现的；另一种是基于 Fast Paxos 算法实现的。系统默认的选举算法为 Fast Paxos。其选举流程如下：

（1）选举线程由当前服务器发起选举的线程担任，其主要功能是对投票结果进行统计，并选出推荐的服务器。

（2）选举线程向集群中的所有服务器发起一次询问（包括自己）。

（3）选举线程收到回复后，验证是否是自己发起的询问（验证 zxid 是否一致），然后获取对方的 id（myid），并存储到当前询问对象列表中，最后获取对方提议的 Leader 相关信息（id,zxid），并将这些信息存储到当次选举的投票记录表中。

（4）收到所有服务器的回复以后，获取 zxid 最大的服务器，并将这台服务器设置成下一次要投票的服务器。

（5）线程将当前 zxid 最大的服务器设置为当前服务器要推荐的 Leader，如果此时获胜



的服务器获得  $n/2+1$  的服务器票数，则设置当前推荐的 Leader 为获胜的服务器，并根据获胜的服务器相关信息设置自己的状态；否则，继续上述过程，直到 Leader 被选举出来。

通过流程分析我们可以得出：要使 Leader 获得大多数服务器的支持，服务器总数必须是奇数  $2n+1$ ，且存活的服务器数目不得少于  $n+1$ 。

每台服务器启动后都会重复以上流程。在恢复模式下，如果是刚从崩溃状态恢复的或者刚启动的服务器，还会从磁盘快照中恢复数据和会话信息，ZooKeeper 会记录事务日志并定期进行快照，方便进行状态恢复。选主的具体流程图如图 3-14 所示。

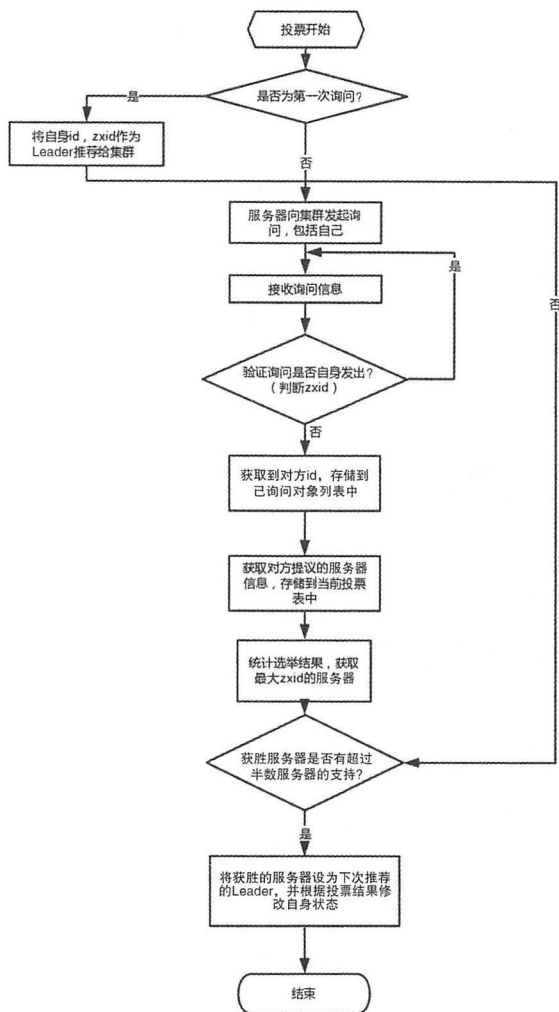


图 3-14





### 3.4.2 Dubbo选型

#### 1. Dubbo背景

随着互联网的发展，网站应用的规模不断扩大，常规的垂直应用架构已无法应对，构建分布式服务架构及流动计算架构势在必行，急需一个治理系统确保架构有条不紊地演进。

#### 2. Dubbo应用

Dubbo 用于大规模服务化，通过在消费方获取服务提供方地址列表，实现软负载均衡，减轻硬件压力。

#### 3. Dubbo架构

Dubbo 架构示意图如图 3-15 所示。

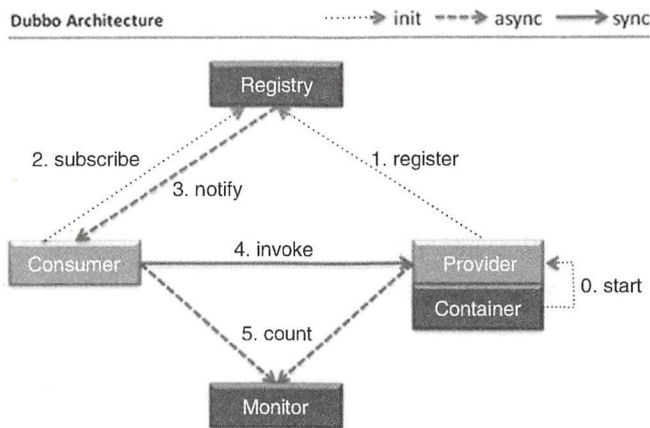


图 3-15

节点角色说明如下。

- ⊙ **Provider**: 暴露服务的提供者。
- ⊙ **Consumer**: 调用远程服务的消费者。
- ⊙ **Registry**: 服务注册与发现的注册中心。



◎ **Monitor:** 统计服务的调用次数和调用时间的监控中心。

◎ **Container:** 服务运行容器。

调用关系说明如下。

0. 服务运行容器负责启动、加载、运行服务提供者。

1. 服务提供者在启动时，向注册中心注册自己提供的服务。

2. 服务消费者在启动时，向注册中心订阅自己所需的服务。

3. 注册中心返回服务提供者地址列表给服务消费者，如果有变更，注册中心将基于长连接推送变更数据给服务消费者。

4. 服务消费者基于软负载均衡算法，从服务提供者地址列表中选一个服务提供者进行调用，如果调用失败，再选另一个服务提供者调用。

5. 服务消费者和提供者在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

## 4. 使用协议

### (1) Dubbo 协议

采用 NIO 复用单一长连接，并使用线程池并发处理请求，减少了握手次数，提高了并发效率，性能较好（推荐使用），但是在传输大文件时单一长连接会成为瓶颈。

Dubbo 协议默认每个服务、每个提供者、每个消费者使用单一长连接，如果数据量较大，则可以使用多个连接。示例如下：

```
<dubbo:protocol name="dubbo" connections="2" />
```

`<dubbo:service connections="0">`或`<dubbo:reference connections="0">`表示该服务使用 JVM 共享长连接（默认）。

`<dubbo:service connections="1">`或`<dubbo:reference connections="1">`表示该服务使用独立的长连接。

`<dubbo:service connections="2">`或`<dubbo:reference connections="2">`表示该服务使用两个独立的长连接。



- ◎ 连接个数：单连接。
- ◎ 连接方式：长连接。
- ◎ 传输协议：TCP。
- ◎ 传输方式：NIO 异步传输。
- ◎ 序列化：Hessian 二进制序列化。
- ◎ 适用范围：传入/传出的参数数据包较小（建议小于 100KB），消费者比提供者个数多，尽量不要用 Dubbo 协议传输大文件或超大字符串。
- ◎ 适用场景：常规的远程服务方法调用。
- ◎ 协议约束：
  - 参数及返回值需要实现 `Serializable` 接口。
  - 参数及返回值不能自定义实现 `List`, `Map`, `Number`, `Date`, `Calendar` 等接口，只能用 JDK 自带的实现，因为 Hessian 会做特殊处理，自定义实现类中的属性值都会丢失。
  - 只传成员属性值和值的类型，不传方法或静态变量。
  - 在接口中增加方法对客户端无影响，如果该方法不是客户端需要的，客户端不需要重新部署；在输入参数和结果集中增加属性对客户端无影响，如果客户端并不需要新属性，则不用重新部署；输入参数和结果集属性名发生变化对客户端序列化无影响，但是如果客户端不重新部署，那么不管是输入还是输出，都获取不到属性名发生变化的属性值。

## （2）RMI 协议

可以与原生 RMI 服务互操作，基于 TCP 协议，偶尔会连接失败，需要重建 Stub。

如果服务接口继承了 `java.rmi.Remote` 接口，则可以和原生 RMI 服务互操作，即：提供者用 Dubbo 的 RMI 协议暴露服务，消费者直接用标准的 RMI 接口调用，或者提供者用标准的 RMI 暴露服务，消费者用 Dubbo 的 RMI 协议调用。

如果服务接口没有继承 `java.rmi.Remote` 接口，默认 Dubbo 将自动生成一个 `com.xxx.XxxService$Remote` 接口，并继承 `java.rmi.Remote` 接口，以此接口暴露服务，但如果设置了 `<dubbo:protocol name="rmi" codec="spring" />`，将不生成 `$Remote` 接口，而是





使用 Spring 的 `RmiInvocationHandler` 接口暴露服务，和 Spring 兼容。

- ◎ 连接个数：多连接。
- ◎ 连接方式：短连接。
- ◎ 传输协议：TCP。
- ◎ 传输方式：同步传输。
- ◎ 序列化：Java 标准二进制序列化。
- ◎ 适用范围：传入/传出的参数数据包大小混合，消费者与提供者个数差不多，可传文件。
- ◎ 适用场景：常规的远程服务方法调用，与原生 RMI 服务互操作。
- ◎ 协议约束：
  - 参数及返回值需要实现 `Serializable` 接口。
  - Dubbo 配置中的超时时间对 RMI 无效，需要使用 Java 启动参数设置 `-Dsun.rmi.transport.tcp.responseTimeout=3000`。

### （3）Hessian 协议

可以与原生 Hessian 服务互操作，基于 HTTP 协议，需要 `hessian.jar` 的支持，HTTP 短连接的开销大。

- ◎ 连接个数：多连接。
- ◎ 连接方式：短连接。
- ◎ 传输协议：HTTP。
- ◎ 传输方式：同步传输。
- ◎ 序列化：Hessian 二进制序列化。
- ◎ 适用范围：传入/传出的参数数据包较大，提供者比消费者个数多，提供者压力较大，可传文件。
- ◎ 适用场景：页面传输、文件传输，或者与原生 Hessian 服务互操作。



◎ 协议约束：

- 参数及返回值需要实现 `Serializable` 接口。
- 参数及返回值不能自定义实现 `List`, `Map`, `Number`, `Date`, `Calendar` 等接口，只能用 JDK 自带的实现，因为 Hessian 会做特殊处理，自定义实现类中的属性值都会丢失。

#### (4) HTTP 协议

基于 HTTP 表单的远程调用协议。

- ◎ 连接个数：多连接。
- ◎ 连接方式：短连接。
- ◎ 传输协议：HTTP。
- ◎ 传输方式：同步传输。
- ◎ 序列化：表单序列化。
- ◎ 适用范围：传入/传出的参数数据包大小混合，提供者比消费者个数多，可用浏览器查看，可用表单或 URL 传入参数，暂不支持传文件。
- ◎ 适用场景：需要同时给应用程序和浏览器 JavaScript 使用的服务。
- ◎ 协议约束：参数及返回值需要符合 Bean 规范。

使用协议示例如下：

```
<dubbo:protocol name="http" port="8080" />
```

注意：如果使用 Servlet 派发请求。

- ◎ 协议的端口`<dubbo:protocol port="8080" />`必须与 Servlet 容器的端口相同。
- ◎ 协议的上下文路径`<dubbo:protocol contextpath="foo" />`必须与 Servlet 应用的上下文路径相同。

#### (5) WebService 协议

基于 CXF 的 `frontend-simple` 和 `transports-http` 实现，可以和原生 WebService 服务互操作，即：提供者用 Dubbo 的 WebService 协议暴露服务，消费者直接用标准的 WebService



接口调用，或者提供者用标准的 WebService 暴露服务，消费者用 Dubbo 的 WebService 协议调用。

- ◎ 连接个数：多连接。
- ◎ 连接方式：短连接。
- ◎ 传输协议：HTTP。
- ◎ 传输方式：同步传输。
- ◎ 序列化：SOAP 文本序列化。
- ◎ 适用场景：系统集成、跨语言调用。
- ◎ 协议约束：
  - 参数及返回值需要实现 Serializable 接口。
  - 参数尽量使用基本类型和 POJO。

## 5. 使用集群

### (1) 特性

- ◎ 注册中心通过长连接感知服务提供者的存在，如果服务提供者宕机，注册中心将立即推送事件通知消费者。
- ◎ 如果注册中心和监控中心全部宕机，则不会影响已运行的提供者和消费者，因为消费者在本地缓存了提供者列表。
- ◎ 注册中心对等集群，当任意一台机器宕机后，将自动切换到另一台机器。
- ◎ 服务提供者无状态，任意一台机器宕机，都不会影响使用。
- ◎ 服务提供者全部宕机后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复。
- ◎ 服务提供者无状态，可动态增加机器部署实例，注册中心将推送新的服务提供者信息给消费者。

### (2) 容错模式

#### -Failover Cluster

- ◎ 失败自动切换，当失败时重试其他服务器（默认）。



- ◎ 通常用于读操作，但重试会带来更大的延迟。
- ◎ 可以通过 `retries="2"` 来设置重试次数（不含第一次）。

#### -Failfast Cluster

- ◎ 快速失败，只发起一次调用，失败立即报错。
- ◎ 通常用于非幂等性的写操作，比如新增记录。

#### -Failsafe Cluster

- ◎ 失败安全，当出现异常时直接忽略。
- ◎ 通常用于写入审计日志等操作。

#### -Failback Cluster

- ◎ 失败自动恢复，后台记录失败请求，定时重发。
- ◎ 通常用于消息通知操作。

#### -Forking Cluster

- ◎ 并行调用多台服务器，只要一个调用成功即返回。
- ◎ 通常用于对实时性要求较高的读操作，但会浪费更多的服务资源。
- ◎ 可以通过 `forks="2"` 来设置最大并行数。

#### -Broadcast Cluster

- ◎ 广播调用所有提供者，逐个调用，任意一个报错则报错（从 Dubbo 2.1.0 版本开始支持）。
- ◎ 通常用于通知所有提供者更新缓存或日志等本地资源信息。

集群模式配置如下：

```
<dubbo:service cluster="failsafe" />
```

或者

```
<dubbo:reference cluster="failsafe" />
```

### （3）负载均衡

#### -Random LoadBalance





- ⊙ 随机，按权重设置随机概率。
- ⊙ 在一个截面上碰撞的概率高，但调用量越大分布越均匀，而且按概率使用权重后负载也比较均匀，有利于动态调整提供者权重。

#### -RoundRobin LoadBalance

- ⊙ 轮询，按公约后的权重设置轮询比率。
- ⊙ 存在慢的提供者累积请求问题，比如第二台机器很慢，但没挂掉，当请求调用到第二台机器时就卡在那里，久而久之，所有请求都卡在第二台机器上。

#### -LeastActive LoadBalance

- ⊙ 最少活跃调用数，相同活跃数的调用随机，活跃数指调用前后计数差。
- ⊙ 慢的提供者收到的请求少，因为越慢的提供者在调用前后计数差会越大。

#### -ConsistentHash LoadBalance

- ⊙ 一致性 Hash，相同参数的请求总是被发往同一个提供者。
- ⊙ 当某一个提供者挂掉时，原本发往该提供者的请求基于虚拟节点被平摊到其他提供者，不会引起剧烈变动。
- ⊙ 算法参见：[http://en.wikipedia.org/wiki/Consistent\\_hashing](http://en.wikipedia.org/wiki/Consistent_hashing)。
- ⊙ 默认只对第一个参数 Hash，如果要修改，请配置 `<dubbo:parameter key="hash.arguments" value="0,1" />`。
- ⊙ 默认用 160 个虚拟节点，如果要修改，请配置 `<dubbo:parameter key="hash.nodes" value="320" />`。

负载均衡配置如下：

```
<dubbo:service interface="..." loadbalance="roundrobin" />
```

或者

```
<dubbo:reference interface="..." loadbalance="roundrobin" />
```

可以在方法中配置：

```
<dubbo:service interface="...">
  <dubbo:method name="..." loadbalance="roundrobin"/>
</dubbo:service>
```



## 6. Dubbo优势

- ◎ 已经成熟，并且被大量项目使用。
- ◎ 开发简单，与正常开发一样，只需额外配置一段 XML（Dubbo 支持注解方式，这样会更简单，此处不讨论）。
- ◎ 可以很容易地切入当前工程中，无耦合。可以以很小的代价在当前的渠道项目中扩展，甚至可以直接使用（无任何修改或者修改很小）原有的接口和方法，并不需要写太多的东西，就可以快速搭建起服务平台。

## 7. Dubbo劣势

- ◎ 网上资料不是很多，在遇到问题时需要上官网查找解决方案。
- ◎ 由于开发过于简捷，隐藏了太多的细节，如有特殊要求，改动麻烦。
- ◎ 基本的一些配置、依赖与优化需要投入精力进行研究。

## 8. 注意事项

- ◎ 单机可以启动多个消费者，但因端口冲突不可以启动多个未经更改的提供者。可以在不同的机器上运行服务提供者，或者修改协议的端口。

```
<dubbo:protocol name="dubbo" port="20880" />
```

- ◎ 如果使用 ZooKeeper 作为注册中心，请注意 JAR 或者 Maven 依赖的冲突问题。
- ◎ 如果参数在多个地方都有配置（如 timeout），则方法级优先，接口级次之，全局配置最后。如果级别一样，则消费者优先，提供者次之。
- ◎ 配置覆盖的策略为：JVM 启动-D 参数优先，XML 次之，Properties 最后。
- ◎ 从 Dubbo 2.2.0 开始，每个服务默认都会在本机暴露；在引用服务时，默认优先引用本地服务；如果希望引用远程服务，则可以通过配置强制引用。例如：

```
<dubbo:reference ... scope="remote" />..
```

### 3.4.3 中间件选型

消息队列中间件（简称“消息中间件”）是指利用高效、可靠的消息传递机制进行与



平台无关的数据交流，并基于数据通信来进行分布式系统的集成。通过提供消息传递和消息排队模型，消息中间件可以在分布式环境下提供应用解耦、弹性伸缩、冗余存储、流量削峰、异步通信、数据同步等功能，其作为分布式系统架构中的一个重要组件，有着举足轻重的地位。

目前开源的消息中间件可谓琳琅满目，大家耳熟能详的就有很多，比如 ActiveMQ、RabbitMQ、Kafka、RocketMQ、ZeroMQ 等。

RocketMQ 的前身为 MetaQ，MetaQ 的第一个版本可以被看成是 LinkedIn 的 Kafka (Scala) 的 Java 版本，并对其增加了事务的支持。RocketMQ 为 MetaQ 3.0，相比于原始的 Kafka，其擅长点除日志收集之外，还增加了诸如 HA、事务等特性，从功能上讲，可以替代传统的大部分消息队列。目前 RocketMQ 已经在阿里巴巴开始大规模应用，并且预计未来会逐渐取代 Notify、Meta 2.x 以前的所有队列。由于 RocketMQ 还比较新，官方没有给出相应的 Benchmark，因此不便于直接对比 RocketMQ 和其他队列。

下面我们看一下消息在 RabbitMQ 中的必由之路。如图 3-16 所示，从 AMQP 协议层面上说：

- ① 消息先从生产者 (Producer) 出发到达交换器 (Exchange)。
- ② 交换器根据路由规则将消息转发到对应的队列 (Queue) 中。
- ③ 消息在队列中进行存储。
- ④ 消费者 (Consumer) 订阅队列并进行消费。

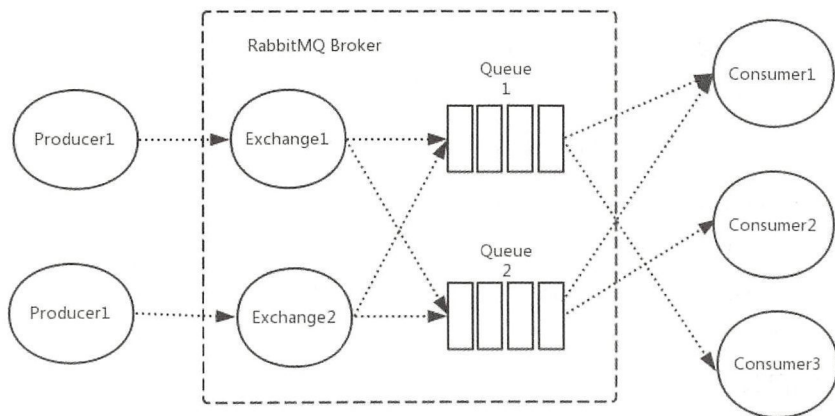


图 3-16

### 3.4.4 Redis

Redis 是一个开源的高性能键值对数据库，它通过各种键值类型来实现不同的数据存储需求。Redis 除了具有存储功能，还可以胜任缓存、队列等不同的角色。

#### 1. 主从复制流程

Redis 包含 Master 和 Slave 两种节点：Master 节点对外提供读写服务；Slave 节点作为 Master 的数据备份，拥有 Master 的全量数据，对外不提供写服务。主从复制由 Slave 主动触发，主要流程如图 3-17 所示。

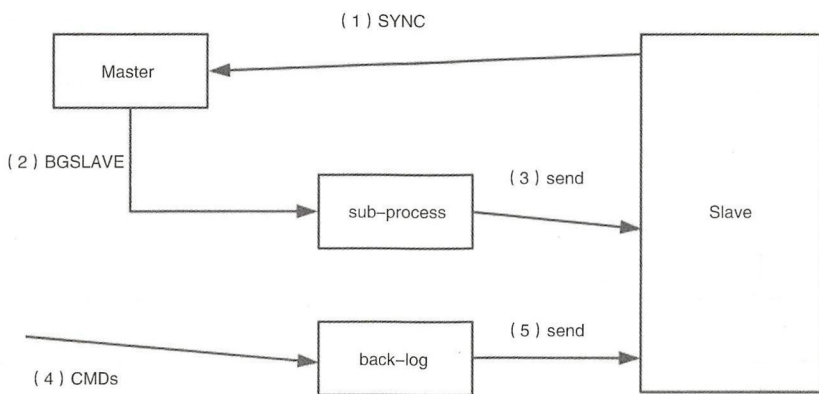


图 3-17

(1) Slave 向 Master 发起 SYNC 命令。这一步在 Slave 启动后触发，Master 被动地将 Slave 节点加入自己的主从复制集群中。

(2) Master 收到 SYNC 命令后，开启 BGSAVE 操作。BGSAVE 是 Redis 的一种全量模式的持久化机制。

(3) 当 BGSAVE 操作完成后，Master 将快照信息发送给 Slave。

(4) 在发送期间，Master 接收到来自客户端的新的写命令，除正常响应之外，再存储一份到 backlog 队列中。

(5) 当快照信息发送完成后，Master 继续发送 backlog 队列信息。

(6) 当 backlog 队列信息发送完成后，后续的写命令同时发送给 Slave，保持实时的异步复制。



## 2. Master故障发现

Sentinel 节点通过定期向 Master 发送心跳包判断其存活状态，称为 PING。一旦发现 Master 没有正确响应，Sentinel 节点就将此 Master 判定为“主观不可用”（这里说的主观，是指“Master 不可用”这个判定尚未得到其他 Sentinel 节点的确认）。随后它将“主观不可用态”发送给其他所有的 Sentinel 节点进行确认（即图 3-18 中的“is-master-down-by-addr”这条交互），当确认的 Sentinel 节点数大于或等于 quorum（可配置）时，则判定该 Master 为“客观不可用”，随后进入 Failover 流程。

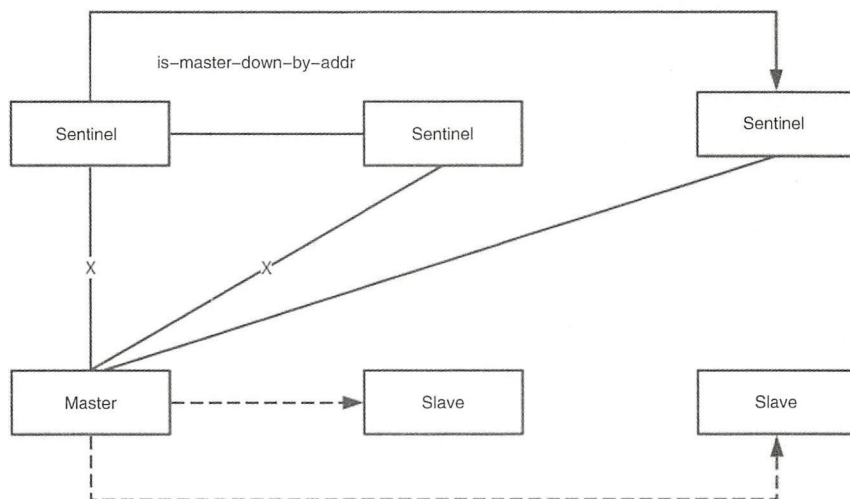


图 3-18

### 3.4.5 数据库

#### 1. MySQL读写分离

读写分离是针对数据库来讲的，随着系统并发量的增大，提高数据访问可用性的一个重要手段就是写数据和读数据分离。当然，在读写分离的同时，需要关注数据的一致性问题；对于一致性问题，在分布式系统 CAP 定理中更多地关注可用性。

#### 2. 高可用方案

我们在设计 MySQL 数据库的高可用架构时，主要应考虑如下几个方面。

- ◎ 如果数据库发生了宕机或意外中断等故障，能尽快恢复数据库的可用性，尽可能减少停机时间，保证业务不会因为数据库故障而中断。
- ◎ 用作备份、只读副本等功能的非主节点的数据应该和主节点的数据实时，或者最终保持一致。
- ◎ 当业务发生数据库切换时，切换前后的数据库内容应当一致，不会因为数据缺失或数据不一致而影响业务。

### (1) 主从或主主半同步复制

使用双节点数据库，搭建单向或者双向的半同步复制环境。在 MySQL 5.7 以后的版本中，由于 Lossless Replication（无损复制）、Logical-clock 并行复制、多线程复制等一系列新特性的引入，使得 MySQL 原生半同步复制更加可靠。常见的架构示意图如图 3-19 所示。

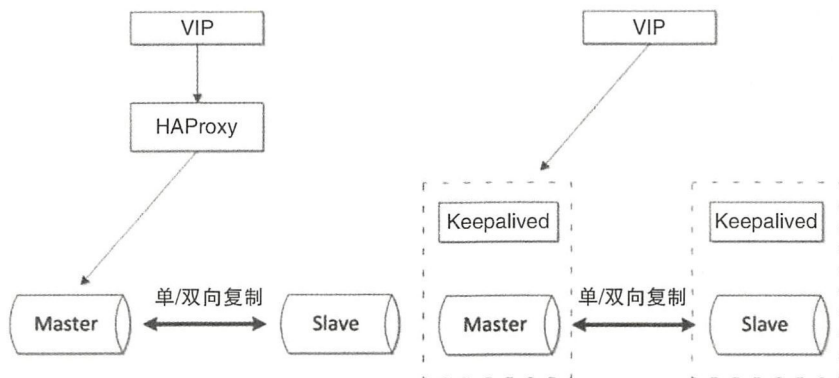


图 3-19

通常和 HAProxy、Keepalived 等第三方软件同时使用，既可以用来监控数据库的健康状况，又可以执行一系列管理命令。如果主库发生故障，那么切换到备库后仍然可以继续使用数据库。

优点如下：

- ◎ 架构比较简单，使用原生半同步复制作为数据同步的依据。
- ◎ 双节点，没有主机宕机后的选主问题，直接切换即可。
- ◎ 双节点，需求资源少，部署简单。

缺点如下：

- ◎ 完全依赖半同步复制，如果半同步复制退化为异步复制，数据一致性将无法得到保证。
- ◎ 需要额外考虑 HAProxy、Keepalived 的高可用机制。

## （2）高可用架构优化

将双节点数据库扩展到多节点数据库，或者多节点数据库集群。可以根据自己的需要选择一主两从、一主多从或者多主多从的集群。

由于半同步复制存在接收到一个从节点的成功应答即认为半同步复制成功的特性，所以多从半同步复制的可靠性要优于单从半同步复制的可靠性。并且多节点同时宕机的概率也要小于单节点宕机的概率，所以在一定程度上可以认为多节点架构的高可用性是好于双节点架构的。

但是由于数据库数量较多，需要数据库管理软件来保证数据库的可维护性。可以选择 MMM、MHA 或者各个版本的 Proxy 等。方案是：MHA+多节点集群，如图 3-20 所示。

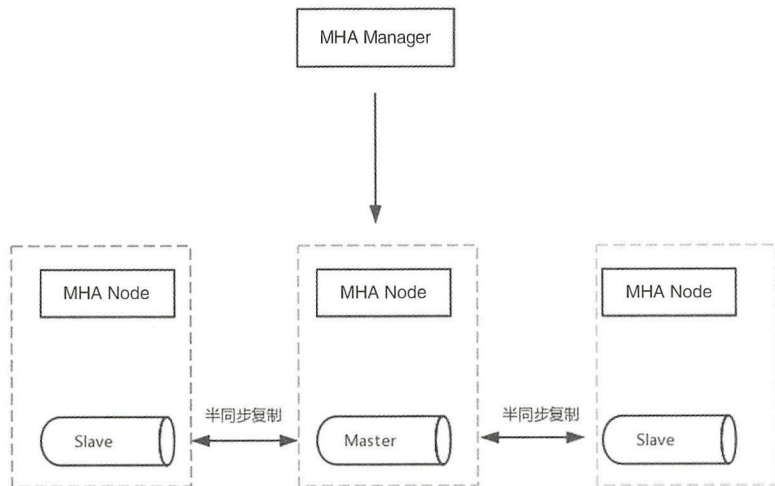


图 3-20

MHA Manager 会定时检测集群中的 Master 节点，当 Master 出现故障时，它可以自动将最新数据的 Slave 提升为新的 Master，然后将所有其他的 Slave 重新指向新的 Master，整个故障转移过程对应用程序完全透明。MHA Node 运行在每台 MySQL 服务器上，其主要作用是在切换时处理二进制日志，以确保尽量少丢数据。

MHA 也可以扩展为多节点集群，如图 3-21 所示。

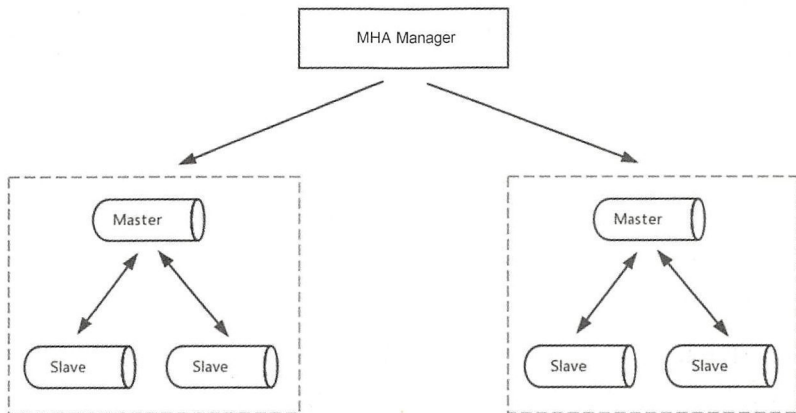


图 3-21

优点如下：

- ◎ 可以进行故障的自动检测和转移。
- ◎ 可扩展性较好，可以根据需要扩展 MySQL 的节点数量和结构。
- ◎ 相比于双节点的 MySQL 复制，三节点/多节点的 MySQL 出现不可用情况的概率更低。

缺点如下：

- ◎ 至少需要三节点，相对于双节点需要更多的资源。
- ◎ 逻辑较为复杂，发生故障后排查问题、定位问题更加困难。
- ◎ 数据一致性仍然靠原生半同步复制来保证，存在数据不一致的风险。
- ◎ 可能因为网络分区发生脑裂现象。

### 3.4.6 MyBatis

MyBatis 是一款优秀的持久层框架，它支持定制化 SQL、存储过程及高级映射。MyBatis 规避了几乎所有的 JDBC 代码和手动设置参数及获取结果集。MyBatis 可以使用简单的 XML 或注解来配置和映射原生信息，将接口和 Java 的 POJO（Plain Old Java Object，简单的 Java 对象）映射成数据库中的记录。



MyBatis 应用程序根据 XML 配置文件创建 `SqlSessionFactory`，`SqlSessionFactory` 再根据配置来源（一是配置文件；二是 Java 代码的注解）获取一个 `SqlSession`。`SqlSession` 包含了执行 SQL 所需要的所有方法，可以通过 `SqlSession` 实例直接运行映射的 SQL 语句，完成对数据的增、删、改、查和事务提交等，用完之后关闭 `SqlSession`。

优点如下：

- ◎ 简单易学。MyBatis 本身就很小且简单，没有任何第三方依赖，最简单的安装是只要两个 JAR 文件+配置几个 SQL 映射文件。通过文档和源代码，就可以完全掌握它的设计思路 and 实现。
- ◎ 灵活。MyBatis 不会对应用程序或者数据库的现有设计强加任何影响。SQL 被写在 XML 里，便于统一管理和优化。通过 SQL 基本上可以实现不使用数据访问框架就可以完成的所有功能。
- ◎ 解除了 SQL 与程序代码的耦合。通过提供 DAL 层，将业务逻辑和数据访问逻辑分离，使系统的设计更清晰、更易维护、更易进行单元测试。
- ◎ 提供映射标签，支持对象与数据库的 ORM 字段关系映射。
- ◎ 提供对象关系映射标签，支持对象关系的组建与维护。
- ◎ 提供 XML 标签，支持编写动态 SQL。

缺点如下：

- ◎ 编写 SQL 语句的工作量很大，尤其是字段多、关联表多时更是如此。
- ◎ SQL 语句依赖数据库，导致数据库移植性差，不能更换数据库。
- ◎ 框架比较简陋，功能尚有缺失，虽然简化了数据绑定代码，但是整个底层数据库查询实际上还需要自己编写代码，工作量也比较大，而且不太容易适应快速数据库修改。
- ◎ 二级缓存机制不佳。

## 1. MyBatis一级缓存

在应用运行过程中，有可能需要在一次数据库会话中执行多次查询条件完全相同的 SQL 语句，为此 MyBatis 提供了一级缓存方案来优化这个场景——如果是相同的 SQL 语

句，则会优先命中一级缓存，避免直接对数据库进行查询，以提高性能。一级缓存执行的时序图如图 3-22 所示。

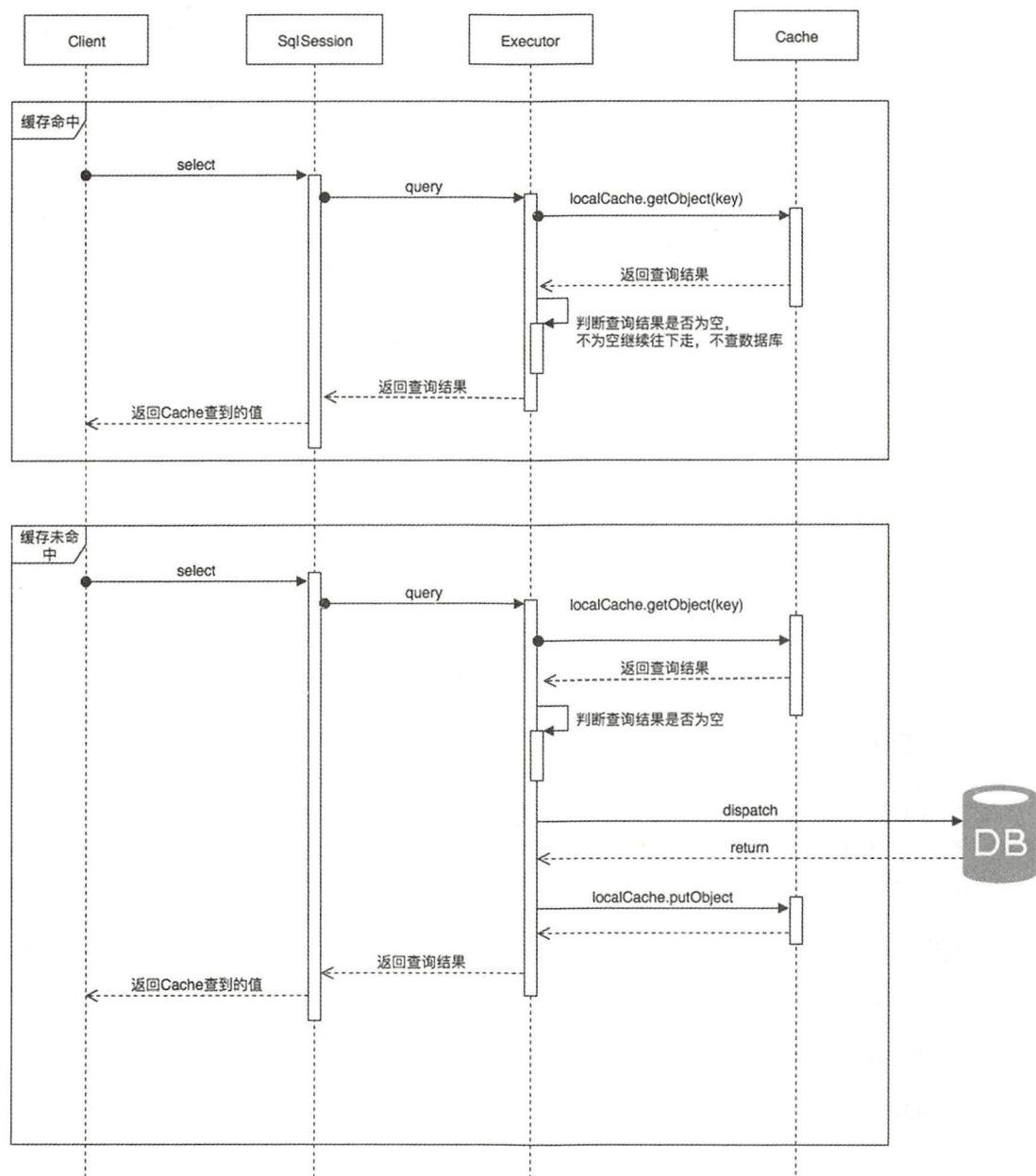


图 3-22

## 2. MyBatis二级缓存

在一级缓存中最大的共享范围就是一个 SqlSession 内部，如果多个 SqlSession 之间需要共享缓存，则需要使用二级缓存。开启二级缓存后，会使用 CachingExecutor 装饰 Executor，在进入一级缓存的查询流程前，先在 CachingExecutor 中进行二级缓存的查询，具体的工作流程如图 3-23 所示。

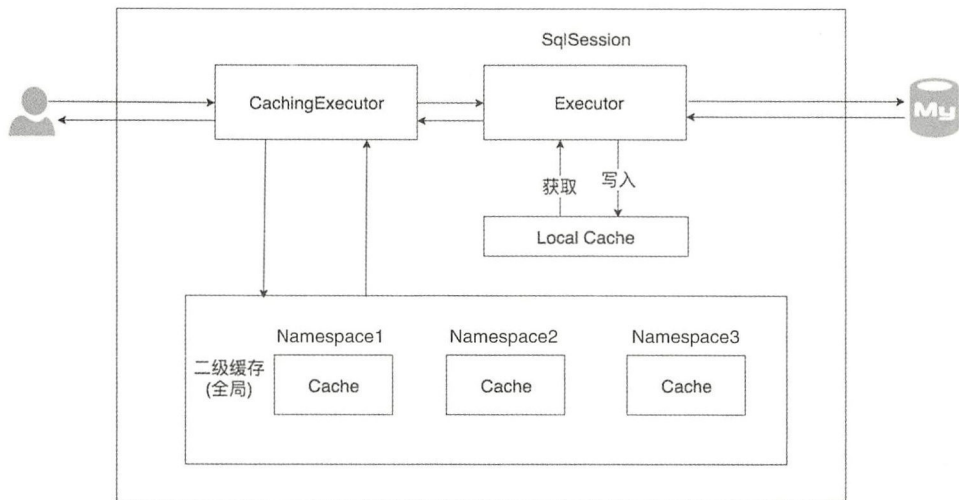


图 3-23

### 3.4.7 Druid

目前 Druid 是最好的数据库连接池，在功能、性能、扩展性方面都超过其他数据库连接池，包括 DBCP、C3P0、BoneCP、Proxool、JBoss DataSource 等。阿里巴巴已经部署了超过 600 个 Druid 应用。

Druid 包含四个部分。

- ⊙ DruidDriver，代理 Driver，能够提供基于 Filter-Chain 模式的插件体系。
- ⊙ DruidDataSource，高效、可管理的数据库连接池。
- ⊙ SQLParser。
- ⊙ 扩展组件。

Druid 的功能如下：

- ◎ 替换 DBCP 和 C3P0。Druid 是一个高效、功能强大、可扩展性好的数据库连接池。
- ◎ 监控数据库访问性能。Druid 内置提供了一个功能强大的 StatFilter 插件，能够详细统计 SQL 的执行性能，这对于线上分析数据库访问性能很有帮助。
- ◎ 数据库密码加密。直接把数据库密码写在配置文件中，这是不好的行为，容易导致安全问题。DruidDriver 和 DruidDataSource 都支持 PasswordCallback。
- ◎ SQL 执行日志。Druid 提供了不同的 LogFilter，能够支持 Common-Logging、Log4j 和 JdkLog，可以根据需要选择相应的 LogFilter，监控应用的数据库访问情况。
- ◎ 扩展 JDBC。如果对 JDBC 层有编程需求，则可以通过 Druid 提供的 Filter 机制，很方便编写 JDBC 层的扩展插件。

### 3.4.8 日志收集

在整个交易过程中会产生大量的日志，这些日志需要被收集到分布式存储系统中存储起来，以便于集中式查询和分析处理。日志收集是大数据的基石。平台每天都会产生大量的日志数据，收集业务日志数据供离线和在线的分析系统使用，正是日志收集系统要做的事情。高可用性、高可靠性和可扩展性是日志收集系统所具有的基本特征。

日志系统需要具备三个基本组件，分别为 Agent（封装数据源，将数据源中的数据发送给 Collector）、Collector（接收多个 Agent 的数据，并进行汇总后导入后端的 Store 中）和 Store（中央存储系统，具有可扩展性和可靠性，支持当前非常流行的 HDFS）。

对于开源的日志收集系统，业界使用得比较多的是 Cloudera 的 Flume 和 Facebook 的 Scribe，目前 FlumeNG 版本对 Flume 在架构上做了较大的改动。

在设计或对日志收集系统做技术选型时，通常需要考虑：

- （1）应用系统和分析系统之间的桥梁，将它们之间的关系解耦
- （2）分布式可扩展

具有高可扩展性，当数据量增加时，可以通过增加节点进行水平扩展。

日志收集系统在各个层次都可伸缩，对数据的处理不需要带状态，在伸缩性方面比较容易实现。



### （3）近实时性

在一些对时效性要求比较高的场景中，需要及时收集日志进行数据分析。

一般的日志文件都会定时或定量进行滚动切割（rolling），所以要实时检测日志文件的生成，及时对日志文件进行类似的 tail 操作，并支持批量发送以提高传输效率。批量发送的时机需要满足消息数量和时间间隔的要求。

### （4）容错性

Scribe 在容错性方面的考虑是，当后端的存储系统崩溃时，Scribe 会将数据写到本地磁盘；当存储系统恢复正常后，Scribe 将日志重新加载到存储系统中。

FlumeNG 通过 Sink Processor 实现负载均衡和故障转移。多个 Sink 可以构成一个 Sink 组。Sink Processor 负责从指定的 Sink 组中激活一个 Sink。Sink Processor 可以通过组中所有的 Sink 实现负载均衡，也可以当一个 Sink 失败时转移到另一个 Sink。

### （5）事务支持

Scribe 没有考虑事务的支持。Flume 通过应答确认机制实现事务的支持，请参见图 3-24。

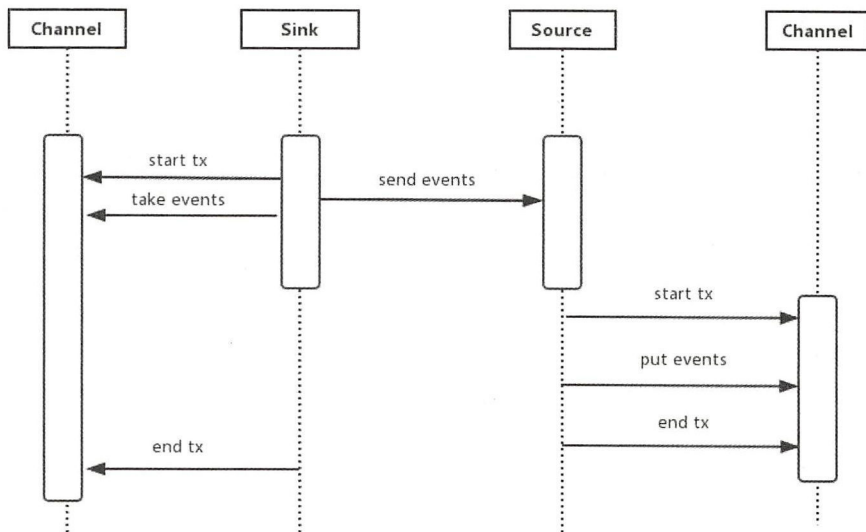


图 3-24

通常提取发送消息都是进行批量操作的，消息确认是对一批数据的确认，这样可以大大提高数据发送的效率。

### （6）可恢复性

FlumeNG 的 Channel 根据对可靠性要求的不同,可以采用基于内存的 Memory Channel 或基于文件的 File Channel。Memory Channel 数据传输的效率比较高,但是当进程死掉、节点宕机后数据丢失,不可恢复;而 File Channel 是一个持久化的隧道,只要磁盘空间足够大,就会将所有事件写入磁盘中,即使宕机也是可以恢复的。

### （7）数据的定时、定量归档

数据经过日志收集系统归集后,一般存储在分布式文件系统如 Hadoop 中。为了便于对数据进行后续的处理分析,需要定时 (TimeTrigger) 或定量 (SizeTrigger) 滚动切割分布式系统的文件。

## 3.4.9 数据同步

在交易系统中,一般需要进行异构数据源的同步,通常将数据文件同步到关系型数据库、将数据文件同步到分布式数据库、将关系型数据库同步到分布式数据库等。数据在异构源之间的同步一般基于性能和业务的需求,将数据存储在本地图文件中一般基于性能的考虑,文件是顺序存储的,效率比较高。将数据同步到关系型数据库一般基于查询的需求;分布式数据库将存储越来越多的海量数据,而关系型数据库无法满足大数据量的存储和查询请求。

在数据同步的设计中,需要综合考虑吞吐量、容错性、可靠性、一致性的问题。同步有实时增量数据同步和离线全量数据同步之分。实时增量数据同步一般使用 tail 实时跟踪文件变化,批量或多线程导出到数据库中。这种方式需要有确认机制,包括两个方面。一个方面是 Channel 需要告知 Agent 确认已经收到数据,发送 LSN 号给 Agent,这样当 Agent 失效恢复时,可以从这个 LSN 点开始 tail 操作。当然,对于允许少量重复记录的问题(当 Channel 向 Agent 发送确认消息时,Agent 宕机并未收到),需要在业务场景中判断。

另一个方面是 sync 告知 Channel 确认已经完成写入数据库的操作,这样 Channel 就可以删除这部分已经确认的消息了。

基于可靠性要求,Channel 可以采用 File Channel 的方式,请参见图 3-25。

离线全量数据同步遵循以空间换取时间、分而治之的原则,尽量缩短数据同步的时间,提高同步的效率。

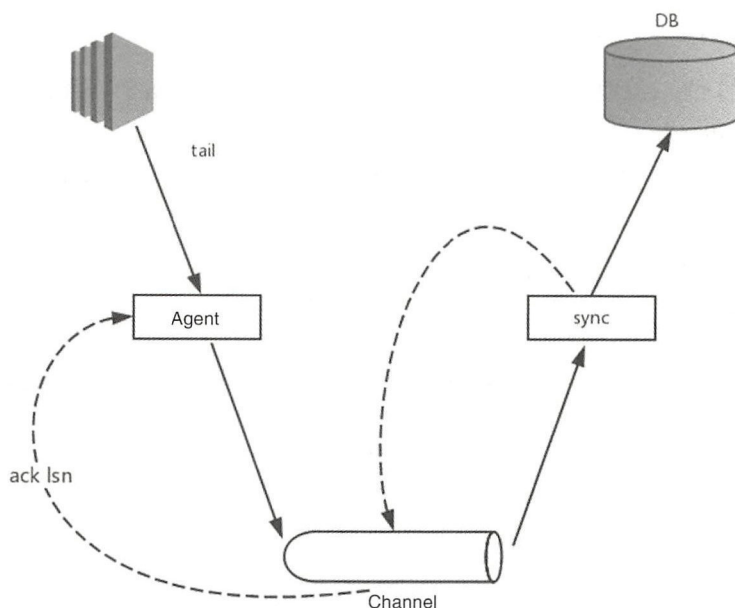


图 3-25

### 3.4.10 数据分析

从传统的基于关系型数据库集群的并行计算，到内存计算，再到基于 Hadoop 等大数据架构进行海量数据批处理，数据分析在大型电子商务网站中应用非常广泛，包括流量统计、推荐引擎、趋势分析、用户行为分析、数据挖掘分类器、分布式索引等。

并行处理集群有 EMC Greenplum。Greenplum 采用的是 MPP（大规模并行处理）架构，是基于 PostgreSQL 开发的大数据量存储的分布式数据库。在内存计算方面有 SAP 的 HANA，也有 Apache 的 Spark。

目前互联网公司主要使用 Hadoop 进行海量数据的离线分析，Hadoop 在可伸缩性、健壮性、计算性能和成本上具有无可替代的优势，事实上它已成为主流的大数据分析平台。

Hadoop 通过 MapReduce 的分布式处理框架处理大规模数据，伸缩性也非常好；但是 MapReduce 最大的不足是不能满足对实时性要求高的场景，其主要用于离线分析。

基于 MapReduce 模型编程进行数据分析，在开发上效率不高，而位于 Hadoop 之上的 Hive 的出现使得数据分析可以以类似于编写 SQL 的方式进行，SQL 经过语法分析、生成

执行计划后，最终生成 MapReduce 任务执行，这样大大提高了开发效率，做到以 Ad-Hoc（即席查询）方式灵活选择查询条件进行统计及分析。

基于 MapReduce 模型的分布式数据分析都是离线分析，在执行上都是暴力扫描，无法利用类似于索引的机制。开源的 Cloudera Impala 是基于 MPP 的并行编程模型的，底层是采用 Hadoop 存储的高性能的实时分析平台，可以大大降低数据分析的延迟。

如果你使用的是 Hadoop 1.0，那么一方面，原有的 MapReduce 框架存在 JobTracker 单点的问题；另一方面，JobTracker 在做资源管理的同时又做任务调度工作，随着数据量的增大和 Job 任务的增多，在可扩展性、内存消耗、线程模型、可靠性和性能上明显存在缺陷、瓶颈。而 Hadoop 2.0 YARN 对整个框架进行了重构，分离了资源管理和任务调度，从架构设计上解决了这个问题。

### 3.4.11 实时计算

在互联网领域，实时计算被广泛应用于实时监控分析、流控、风险控制等场景中。电商平台系统或应用需要对日常产生的大量日志和异常信息进行实时过滤、分析，以判定是否需要预警。同时需要对系统进行自我保护，比如对模块进行流量控制，以防止非预期的压力过大而引起系统瘫痪，当流量过大时，可以采取拒绝或引流等机制。另外，有些业务需要进行风险控制，比如有些彩票业务需要根据系统的实时销售情况进行限号与放号等。

随着系统信息量的爆炸式产生及计算复杂度的增加，基于单节点的计算已不能满足实时计算的要求，需要进行多节点的分布式计算，为此就出现了分布式实时计算平台。

这里所说的实时计算其实是指流式计算，其概念前身是 CEP（复杂事件处理），相关的开源产品有 Esper，以及分布式流计算产品 Yahoo S4、Twitter Storm 等，Storm 开源产品的使用最广泛。

对于实时计算平台，在架构设计上需要考虑以下几个因素。

- ◎ 伸缩性。随着业务量和计算量的增加，通过增加节点就可以处理。
- ◎ 高性能、低延迟。从数据流入计算平台，到计算输出结果，需要高效的性能且低延迟，保证消息得到快速处理，做到实时计算。
- ◎ 可靠性。保证每条数据消息都会得到一次完整处理。
- ◎ 容错性。系统可以自动管理节点的宕机失效，这对应用来说是透明的。



Twitter Storm 在以上几个方面做得比较好，下面简单介绍 Storm 的架构，如图 3-26 所示。

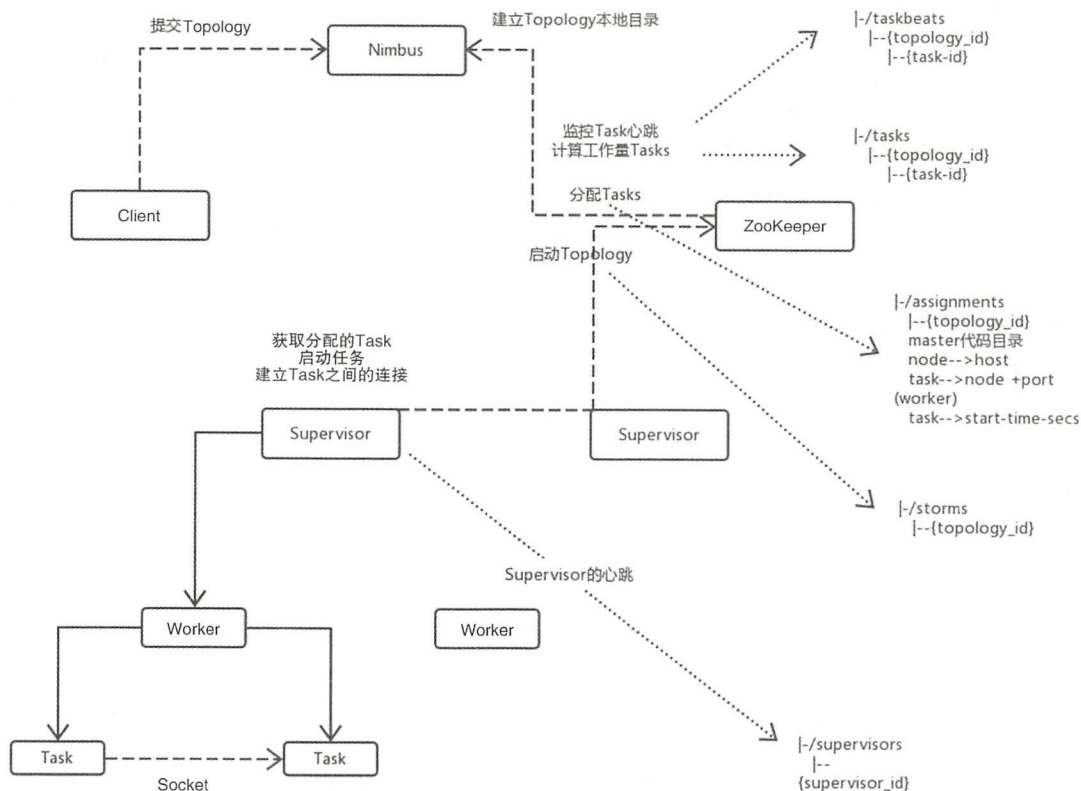


图 3-26

整个集群是通过 ZooKeeper 来进行管理的。

客户端提交 Topology 到 Nimbus。Nimbus 针对该拓扑建立本地目录，根据 Topology 的配置计算 Task、分配 Task，在 ZooKeeper 上建立 Assignments 节点存储 Task 和 Supervisor 机器节点中 Worker 的对应关系。

在 ZooKeeper 上创建 Taskbeats 节点来监控 Task 的心跳，启动 Topology。

Supervisor 到 ZooKeeper 上获取所分配的 Task，启动多个 Worker，每个 Worker 都生成一个 Task，一个 Task 一个线程；根据 Topology 信息初始化建立 Task 之间的连接；Task 和 Task 之间是通过 ZeroMQ 通信的；之后整个 Topology 运行起来。

Tuple 是流的基本处理单元，也就是一条消息，Tuple 在 Task 中流转。Tuple 的发送和

接收过程如下：

(1) 发送 Tuple。Worker 提供了一个转发 (Transfer) 功能，用于当前 Task 把 Tuple 发送到其他 Task 中。发送方将 Tuple 数据序列化后放到转发队列中，接收方仅序列化获得 Tuple 数据。

在 Storm 0.8 版本之前，这个队列是 `LinkedBlockingQueue`，在 Storm 0.8 版本之后是 `DisruptorQueue`。

在 Storm 0.8 版本之后，每一个 Worker 都绑定一个 Inbound Queue 和 Outbound Queue，其中 Inbound Queue 用于接收消息，Outbound Queue 用于发送消息。

在发送消息时，由单个线程从转发队列中拉取数据，把这个 Tuple 通过 ZeroMQ 发送到其他 Worker 中。

(2) 接收 Tuple。每一个 Worker 都会通过监听 ZeroMQ 的 TCP 端口来接收消息，将消息放到 `DisruptorQueue` 中后，从队列中获取消息 (taskid,tuple)，根据目的 taskid,tuple 的值路由到 Task 中执行。每一个 Tuple 都可以被发送到 Direct Stream 中，也可以被发送到 Regular Stream 中。

通过以上分析可以看到，Storm 在伸缩性、容错性、高性能方面的特性，从架构设计层面得到强有力的支撑；在可靠性方面，Storm 的 ack 组件利用 XOR (异或) 算法，在不失性能的同时，保证每一条消息都能得到完整的处理。

### 3.4.12 实时推送

实时推送的应用场景非常多，比如动态监控系统的实时曲线绘制、手机消息推送、Web 实时聊天等。

实时推送有很多技术可以实现，如 Comet、WebSocket 等。

Comet 基于服务器长连接的“服务器推”技术，有两种实现模型。

(1) HTTP Long Polling，基于 AJAX 的长轮询方式。它与传统 AJAX 应用有些不同，服务器端会阻塞请求直到有数据返回或超时，并且客户端 JavaScript 响应函数在处理完服务端返回的信息后，会立即发出请求重新建立连接。

(2) HTTP Stream：基于 iframe 和 htmlfile 的流方式。通过在 HTML 页面中嵌入一个隐藏帧，然后将这个隐藏帧的 SRC 属性设置为对一个长连接的请求，服务器端就能源源

不断地往客户端输入数据。

WebSocket 是从 HTML 5 开始提供的一种在单个 TCP 连接上进行全双工通信的协议，它实现了浏览器与服务器的双向通信。在 WebSocket API 中，浏览器和服务器只需要通过一个握手的动作，便能在浏览器与客户端之间形成快速的双向通道，使得数据可以快速地双向传播。Socket.io 是一个 Node.js WebSocket 库，包括客户端的 JavaScript 和服务端端的 Node.js，用于快速构建实时的 Web 应用。

## 3.5 本章小结

在这一章中，讲述了交易系统的设计理念、架构技术及业务功能，并从技术角度介绍了当今流行的技术框架。框架没有最好的，只有最适合的。只有从自己产品的业务特点、数据规模、数据特性入手进行分析，经过不断尝试、不断优化，才能找到最适合自己的产品的框架。

# 4

## 第 4 章

## 交易系统功能

### 4.1 前台功能

#### 4.1.1 交易

##### 1. 交易行情K线

如图 4-1 所示为交易行情 K 线图。

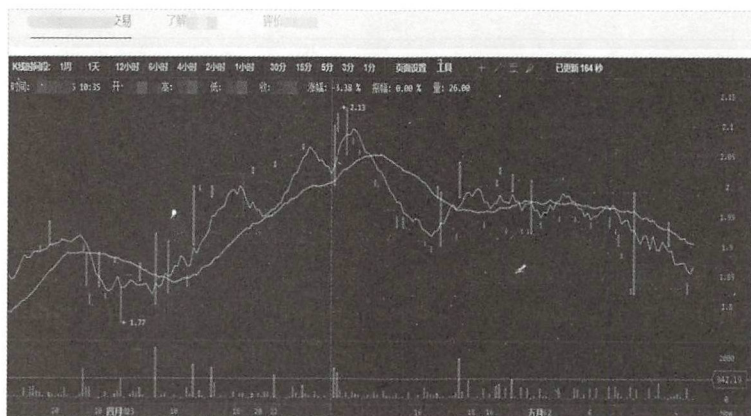


图 4-1



功能说明：

- ◎ 将鼠标指针在 K 线界面上移动，可显示每个时间点的开盘价、最高价、最低价、收盘价、涨幅、振幅和成交量。
- ◎ 点击最上方的 1 周、1 天、12 小时、6 小时、4 小时、2 小时、1 小时、30 分、15 分、5 分、3 分和 1 分，可查看相应时间频率下的交易数据。

与 K 线相关的数据如下。

### (1) K 线数据格式

线性图数据格式：

```
[
  [
    1416182400000, 时间戳
    7.204, 开盘价
    7.232, 最高价
    7.037, 最低价
    7.072, 收盘价
  ],
  [
    1416268800000, 时间戳
    7.058, 开盘价
    7.107, 最高价
    6.933, 最低价
    6.968, 收盘价
  ],
]
```

普通数据、成交量和均线数据格式：

```
[
  [
    1416182400000, 时间戳
    7.204, 价格
  ],
  [
    1416268800000, 时间戳
    7.058, 价格
  ],
]
```

## (2) 与 K 线相关的 SQL 语句

### ① 买入 SQL 语句。

获得委托买入信息，根据指定市场 ID{scid}，查询交易中、允许撮合、买入交易的委托信息，按委托价格分组计算未成交总数量及总金额，返回委托价格最高的 10 条记录。

```
SELECT
    marketId AS scid,
    entrustPrice AS price,
    SUM(entrustCount - dealCount) AS dealCount,
    SUM(
        (entrustCount - dealCount) * entrustPrice
    ) balance
FROM
    trust_datas
WHERE
    type = 'JYZ'
    AND isMatched = 'S'
    AND tradeType = 'MR'
    AND marketId = #{scid}
GROUP BY
    marketId,
    entrustPrice,
    tradeType
ORDER BY
    entrustPrice DESC
LIMIT 10
```

### ② 卖出 SQL 语句。

获得委托卖出信息，根据指定市场 ID{scid}，查询交易中、允许撮合、卖出交易的委托信息，按委托价格分组计算未成交总数量及总金额，返回委托价格最低的 10 条记录。

```
SELECT
    marketId AS scid,
    entrustPrice AS price,
    SUM(entrustCount - dealCount) AS dealCount,
    SUM(
        (entrustCount - dealCount) * entrustPrice
    ) balance
FROM
```

```

    trust_datas
WHERE
    type = 'JYZ'
    AND isMatched = 'S'
    AND tradeType = 'MC'
    AND marketId = #{scid}
GROUP BY
    marketId,
    entrustPrice,
    tradeType
ORDER BY
    entrustPrice DESC
LIMIT 10

```

### ③K 线图 SQL 语句。

时间维度为 1 分钟、3 分钟、5 分钟、15 分钟、30 分钟。根据指定市场 ID{scid}及时间维度{time}（单位：秒），查询获得最新 480 条 JSON 格式的数据。

JSON 数据字段：K 线时间，0，0，开盘价，收盘价，最高价，最低价，成交量。

```

SELECT
    kLines_data
FROM
    kLines_ #{scid}_#{time}
WHERE
    marketId = #{scid}
    AND timeType = #{time}
ORDER BY
    id DESC
LIMIT 480

```

时间维度为 7 天、1 天、12 小时、6 小时。根据指定市场 ID{scid}及时间维度{time}（单位：秒），查询获得最新 600 条 JSON 格式的数据。

JSON 数据字段：K 线时间，0，0，开盘价，收盘价，最高价，最低价，成交量。

```

SELECT
    F10
FROM
    kLines_market_n
WHERE
    marketId = #{scid}

```



```

        AND timeType = #{time}
ORDER BY
    id DESC
LIMIT 600

```

时间维度为 1 小时、2 小时、4 小时。根据指定市场 ID{scid} 及时间维度 {time} (单位: 秒), 查询获得最新 600 条 JSON 格式的数据。

JSON 数据字段: K 线时间, 0, 0, 开盘价, 收盘价, 最高价, 最低价, 成交量。

```

SELECT
    F10
FROM
    kLines_market_m
WHERE
    marketId = #{scid}
    AND timeType = #{time}
ORDER BY
    id DESC
LIMIT 600

```

### (3) K 线定时器与数据组装

#### ① 行情。

- ◎ 对昨日收盘价、当前价、最高价、最低价、买一价、卖一价、成交量、成交额进行补零操作。
- ◎ 如果成交量和成交额数值超过 10000, 单位改成“万”, 计算日涨跌。

#### ② 查询 K 线图最新记录。

数据格式: market+ 市场 ID+since0/since1:List<KlineEntityNew> (List 表示数据库直接查询)。

```

/**
 * @查询 K 线图最新记录
 */
public void getKlineNew() throws Exception {
    if (LineData.scid == null) {
        throw new Exception("市场不存在。");
    }
    for (String since : LineData.since.keySet()) {

```



## 区块链：交易系统开发指南

```

        for (String market : LineData.market.keySet()) {
            if (LineData.since.get(since) <= 0) {
                LineData.klineEntityNew.put(market + since,
indexDao.klineEntityNew1(LineData.market.get(market)));
            } else {
                // 根据指定市场 ID{scid}及指定起始成交 ID{since},
                // 查询获得最新一条成交记录: 成交 ID, 成交时间, 成交价格, 成交数量
                LineData.klineEntityNew.put(market + since,
indexDao.klineEntityNew2(LineData.market.get(market),
LineData.since.get(since)));
            }
        }
    }
}

```

## ③查询 K 线图历史记录: 1 分钟、3 分钟、5 分钟、15 分钟、30 分钟。

```

/**
 * @查询 K 线图历史记录
 */
public void getKlineHistory1() {
    Set<Entry<String, Long>> markets = LineData.market.entrySet();
    Set<Entry<String, Long>> time = LineData.time.entrySet();
    for (Entry<String, Long> market : markets) {
        for (Entry<String, Long> entry : time) {
            Long market1 = market.getValue();
            Long time1 = entry.getValue();
            if (time1 < 3600) {
                List<String> klineHistory1 = indexDao.klineHistory1(market1,
time1);
                if (klineHistory1.size() != 0) {
                    List<String> klineHistory = new ArrayList<>();
                    for (int i = klineHistory1.size() - 1; i >= 0; i--) {
                        klineHistory.add(klineHistory1.get(i));
                    }
                    LineData.klineHistory.put("market" + market1 + "time" +
time1, klineHistory);
                }
            }
        }
    }
}

```

## ④查询 K 线图历史记录：4 小时、2 小时、1 小时。

```

/**
 * @查询 K 线图历史记录
 */
public void getKlineHistory2() {
    Set<Entry<String, Long>> markets = LineData.market.entrySet();
    Set<Entry<String, Long>> time = LineData.time.entrySet();
    for (Entry<String, Long> market : markets) {
        for (Entry<String, Long> entry : time) {
            Long market1 = market.getValue();
            Long time1 = entry.getValue();
            if (time1 >= 60 * 60 && time1 < 360 * 60) {
                List<String> klineHistory2 = indexDao.klineHistory2 (market1, time1);
                if (klineHistory2.size() != 0) {
                    List<String> klineHistory = new ArrayList<>();
                    for (int i = klineHistory2.size() - 1; i >= 0; i--) {
                        klineHistory.add(klineHistory2.get(i));
                    }
                    LineData.klineHistory.put("market" + market1 + "time" +
                        time1, klineHistory);
                }
            }
        }
    }
}

```

## ⑤查询 K 线图历史记录：7 天、1 天、12 小时、6 小时，位数，加载市场 ID、sinceid，5 分钟一次。

```

/**
 * @查询 K 线图历史记录
 */
public void getKlineHistory3() {
    Set<Entry<String, Long>> markets = LineData.market.entrySet();
    Set<Entry<String, Long>> time = LineData.time.entrySet();
    for (Entry<String, Long> market : markets) {
        for (Entry<String, Long> entry : time) {
            Long market1 = market.getValue();
            Long time1 = entry.getValue();
            if (time1 >= 360 * 60) {

```



⑥读取历史 K 线数据。

• 106 •

```

    }
}
if (LineData.klineHistory.get("market" + market + "time" + step) != null) {
    res.getWriter().write(LineData.klineHistory.get("market" + market +
"time" + step).toString());
}
}
}

```

⑦K 线数据的底层 MySQL 存储过程。

KLINE\_ONE\_HOUR 事件调用：CALL SP\_T6018\_KLINE\_1\_HOUR()。

```

CALL SP_T6018_KLINE_1_HOUR():
BEGIN
    START TRANSACTION;
    CALL SP_T6018_KLINE(3600,1);
    CALL SP_T6018_KLINE(3600,2);
    CALL SP_T6018_KLINE(3600,3);
    CALL SP_T6018_KLINE(3600,4);
    CALL SP_T6018_KLINE(3600,5);
    CALL SP_T6018_KLINE(3600,6);
    CALL SP_T6018_KLINE(3600,7);
    CALL SP_T6018_KLINE(3600,8);
    COMMIT;
END

```

SP\_T6018\_KLINE 函数如下：

```

BEGIN
-- 将市场 ID 直接写到存储过程中，这个数据是经常被查询但又不会改变的
-- 传递参数  -时间  -市场
DECLARE _F02          DECIMAL(20,8)          DEFAULT 0; -- 开盘价
DECLARE _F03          DECIMAL(20,8)          DEFAULT 0; -- 收盘价
DECLARE _F04          DECIMAL(20,8)          DEFAULT 0; -- 最高价
DECLARE _F05          DECIMAL(20,8)          DEFAULT 0; -- 最低价
DECLARE _F06          DECIMAL(20,8)          DEFAULT 0; -- 收盘成交额
DECLARE _F07          INT UNSIGNED           DEFAULT 0; -- K 线时间
DECLARE _F08          INT UNSIGNED           DEFAULT 0; -- 创建时间
DECLARE _F09          INT UNSIGNED           DEFAULT 0; -- 成交量
DECLARE _F10          LONGTEXT               DEFAULT ''; -- JSON 数据
DECLARE _IS_DATA INT DEFAULT 0; -- 当前时间段是否有数据
DECLARE _TIME_MINUTE INT DEFAULT 0; -- 得出分钟
DECLARE _TABLE        LONGTEXT              DEFAULT ''; -- 表名

```



## 区块链：交易系统开发指南

```

DECLARE _SQLCOUNTS LONGTEXT DEFAULT ''; -- SQL 语句
DECLARE _TIME_NOW_COUNT_START TIMESTAMP DEFAULT NULL; -- 根据时间条件计算距离当前时间得出相对应的开始时间
DECLARE _TIME_NOW_COUNT_END TIMESTAMP DEFAULT NULL; -- 根据时间条件计算距离当前时间得出相对应的结束时间
DECLARE _COMMENT_NAME VARCHAR(80) DEFAULT '';
DECLARE _TIME_OLD INT UNSIGNED DEFAULT 0; -- 旧时间
DECLARE _TIME_NEW INT UNSIGNED DEFAULT 0; -- 新时间
SET _TIME_MINUTE = _TIME/60;
SET _TIME_NOW_COUNT_START=DATE_FORMAT(DATE_SUB(NOW(),INTERVAL _TIME_MINUTE MINUTE),'%Y-%m-%d %H:%i:00');

SET _TIME_OLD = UNIX_TIMESTAMP(_TIME_NOW_COUNT_START);
SET _TIME_NEW = UNIX_TIMESTAMP(NOW());
SET _TIME_NOW_COUNT_END = DATE_FORMAT(NOW(),'%Y-%m-%d %H:%i:00');

-- 设置表名称
IF _TIME_MINUTE >= 60 AND _TIME_MINUTE < 360 THEN -- 4 小时、2 小时、1 小时插入一个表
    SET _COMMENT_NAME = "K 线交易_4 小时_2 小时_1 小时";
    SET _TABLE = CONCAT("zch_s60.t6019_market_n");
ELSEIF _TIME_MINUTE >= 360 THEN -- 表数据划分为 1 周、1 天、12 小时、6 小时
    SET _COMMENT_NAME = "K 线交易_1 周_1 天_12 小时_6 小时";
    SET _TABLE = CONCAT("zch_s60.t6019_market_m");
ELSE
    SET _TABLE = CONCAT("zch_s60.t6019_",_MARKET,'_',_TIME);
    SET _COMMENT_NAME =CONCAT("K 线交易_",_MARKET,"_",_TIME,"分钟");
END IF;
SET _SQLCOUNTS=CONCAT("
Create Table If Not Exists ", _TABLE,"(
    F01          bigint not null auto_increment comment '编号',
    F02          decimal(20,8) comment '开盘价',
    F03          decimal(20,8) comment '收盘价',
    F04          decimal(20,8) comment '最高价',
    F05          decimal(20,8) comment '最低价',
    F06          decimal(20,8) comment '收盘成交额',
    F07          bigint comment 'K 线时间',
    F08          bigint comment '创建时间',
    F09          bigint comment '成交量',
    F10          text comment 'JSON 数据',
    F11          int comment '时间类型',
    F12          bigint comment '市场编号',

```

```

    primary key (F01),
    KEY `F02` (`F12`,`F11`) USING BTREE
)Engine MyISAM;");

SET @sqlstr_2 = _SQLCOUNTS;
PREPARE stmt1_2 FROM @sqlstr_2 ;
EXECUTE stmt1_2 ;

-- 添加备注, 添加索引
SET _SQLCOUNTS=CONCAT("alter table ", _TABLE," comment '", _COMMENT_NAME,"';");
SET @sqlstr_2 = _SQLCOUNTS;
PREPARE stmt1_2 FROM @sqlstr_2 ;
EXECUTE stmt1_2 ;

SET _SQLCOUNTS=CONCAT('SELECT IFNULL(F01,0) INTO @LENGTH FROM ', _TABLE, '
WHERE F12 = ', _MARKET, ' AND F07 = ', _TIME_OLD, ' LIMIT 1');
SET @sqlstr_1 = _SQLCOUNTS;
PREPARE stmt1_1 FROM @sqlstr_1 ;
EXECUTE stmt1_1 ;

IF @LENGTH <= 0 || @LENGTH is null THEN
-- 判断区间内是否存在数据
-- 判断当前查询时间段是否有数据
SET _IS_DATA = (SELECT IFNULL(F01,0) FROM zch_s60.t6018 WHERE (F11 BETWEEN
_TIME_NOW_COUNT_START AND _TIME_NOW_COUNT_END) AND F02 = _MARKET LIMIT 1);

IF _IS_DATA > 0 THEN
-- 查询开盘价
SET _F02 = (SELECT F05 FROM zch_s60.t6018 WHERE (F11 BETWEEN _TIME_NOW_
COUNT_START AND _TIME_NOW_COUNT_END) AND F02 = _MARKET ORDER BY F11 ASC LIMIT
1);

-- 查询收盘价
SET _F03 = (SELECT F05 FROM zch_s60.t6018 WHERE (F11 BETWEEN _TIME_NOW_
COUNT_START AND _TIME_NOW_COUNT_END) AND F02 = _MARKET ORDER BY F11 DESC LIMIT
1);

-- 查询最高价
SET _F04 = (SELECT MAX(F05) FROM zch_s60.t6018 WHERE (F11 BETWEEN
_TIME_NOW_COUNT_START AND _TIME_NOW_COUNT_END) AND F02 = _MARKET );
-- 查询最低价

```



## 区块链：交易系统开发指南

```

SET _F05 = (SELECT MIN(F05) FROM zch_s60.t6018 WHERE (F11 BETWEEN
_TIME_NOW_COUNT_START AND _TIME_NOW_COUNT_END) AND F02 = _MARKET );

-- 查询收盘成交额
SET _F06 = (SELECT SUM(F07) FROM zch_s60.t6018 WHERE (F11 BETWEEN
_TIME_NOW_COUNT_START AND _TIME_NOW_COUNT_END) AND F02 = _MARKET );

-- K 线时间
SET _F07 = _TIME_OLD;
-- 创建时间
SET _F08 = _TIME_NEW;
-- 成交量
SET _F09 = (SELECT SUM(F06) FROM zch_s60.t6018 WHERE (F11 BETWEEN
_TIME_NOW_COUNT_START AND _TIME_NOW_COUNT_END) AND F02 = _MARKET );
SET _F10 = CONCAT('[',_TIME_OLD,',',0,',',0,',',_F02,',',_F03,',',_F04,',',_F05,',',_F09,']');
-- SET _F10 = CONCAT("{",_TIME_OLD,',',0,',',0,',',_F02,',',_F03,',',_F04,',',_F05,',',_F09,'}');
SET _SQLCOUNTS = CONCAT("INSERT INTO ",_TABLE,' SET F02=',_F02,',F03=',_F03,',F04=',_F04,',F05=',_F05,',F06=',_F06,',F07=',_F07,',F08=',_F08,',F09=',_F09,',F10=',_F10,',F11=',_TIME ',F12=',_MARKET);
SET @sqlstr = _SQLCOUNTS;
PREPARE stmt1 FROM @sqlstr ;
EXECUTE stmt1 ;
END IF;
END IF;
END

```

## 2. 委托

如图 4-2 所示是用户的挂单信息，即委托的买卖数据。

委托记录			
买/卖	价格	数量	总额
卖5	0.120	680.319	81.638
卖4	0.025	7644.680	191.117
卖3	0.014	1500.000	21.000
卖2	0.012	8003.084	96.037
卖1	0.010	17142.667	171.427
买1	0.004	10467.680	41.868
买2	0.003	5555.900	16.668

图 4-2

功能说明：

- ◎ 当输入的买卖价格、买卖数量符合规则时，将产生挂单，并冻结挂单产生的资产金额。但不会立马成交，只有当有合适的买单、卖单出现时，才会交易成功。
- ◎ 在成交之前，若用户不想以挂出的价格交易，则可直接点击“撤销”，委托将被删除，被冻结的法币或虚拟币金额会解冻，回到可用余额中。
- ◎ 在没确定好价格时，最好不要随意挂单买卖，防止交易自动完成，造成资产的损失。

买入、卖出委托代码如下：

```

/ **
* @买入委托信息
* @throws Exception
*/
public Map<String, Object> wd_mr(HttpServletRequest req,
HttpServletResponse res, Model model) throws Exception {
    HttpUtils.setHeader(req, res);
    Map<String, Object> result = new HashMap<>();
    String scid = req.getParameter("scid");
    int Intscid = Integer.parseInt(scid);
    if (Intscid <= 0) {
        result.put("msg", "市场异常");
    }
    if (LineData.getWtMr.get("scid" + Intscid) == null) {
        indexservice.getWtMr();
    }
    result.put("num", "0");
    result.put("msg", "");
    if (LineData.getWtMr != null && LineData.getWtMr.size() > 0) {
        result.put("num", "1");
        result.put("msg", LineData.getWtMr.get("scid" + Intscid));
    }
    return result;
}
/ **
* 卖出委托信息
* @throws Exception

```



```

    */
    public Map<String, Object> wd_mc(HttpServletRequest req,
    HttpServletResponse res, Model model) throws Exception {
        HttpUtils.setHeader(req, res);
        Map<String, Object> result = new HashMap<>();
        String scid = req.getParameter("scid");
        int Intscid = Integer.parseInt(scid);
        if (Intscid <= 0) {
            result.put("msg", "市场异常");
        }
        if (LineData.getWtMc.get("scid" + Intscid) == null) {
            indexservice.getWtMc();
        }
        result.put("num", "0");
        result.put("msg", "");
        if (LineData.getWtMc != null && LineData.getWtMc.size() > 0) {
            result.put("num", "1");
            result.put("msg", LineData.getWtMc.get("scid" + Intscid));
        }
        return result;
    }
}

```

当买入、卖出都有委托挂单时，将执行撮合业务，交易成功的条件是卖方的委托价格不高于买方的价格，即卖方队列与买方队列必须有交集才能促成交易。交易撮合流程图如图 4-3 所示。

交易原则如下：

- ① 高买低卖。
- ② 价格优先。
- ③ 对于同一价格时间优先。
- ④ 先有卖单，委托买单时，买单从低往高查卖单，找到合适的卖单即交易；先有买单，委托卖单时，卖单从高往低查买单，找到合适的买单即交易。

### 3. 买入、卖出

如图 4-4 所示为买入、卖出操作界面。

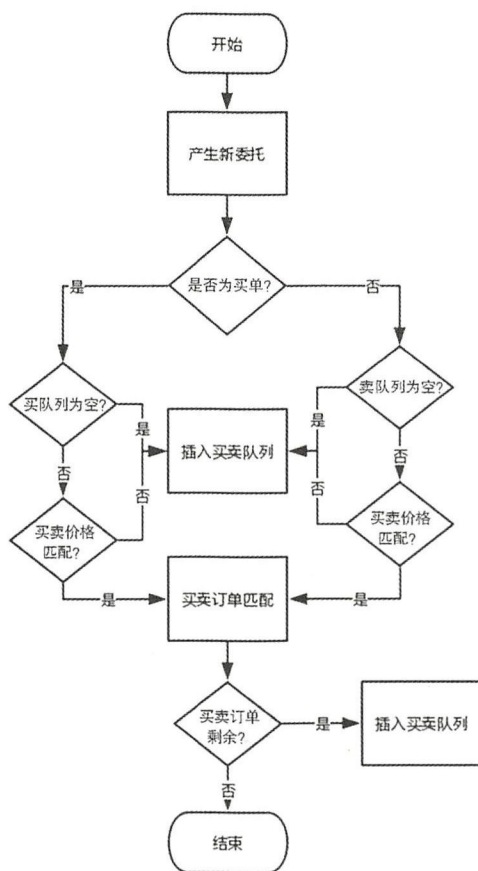


图 4-3

买入	卖出
最佳买价: 0.0165BTC/USD	最佳卖价: 0.0165BTC/USD
买入价格: 此出价高于1个市的价格	卖出价格: 此出价高于1个市的价格
最大可买: 5BTC	最大可卖: 0.0005BTC
买入比例: 1/4 1/3 1/2 全部	卖出比例: 1/4 1/3 1/2 全部
买入数量:	卖出数量:
总价: USD	总价: USD
手续费: 0.100%	手续费: 0.100%
交易密码:	交易密码:
<input type="button" value="买入"/>	<input type="button" value="卖出"/>

图 4-4

功能说明：

- ⊙ 用户在此处添加适当的买入、卖出价格和数量，点击“买入”或“卖出”按钮，就会生成相应的买入、卖出挂单信息。
- ⊙ 若想立马完成交易，买入价格必须高于卖单里的最低价格，卖出价格必须低于买单里的最高价格。

交易中心买入代码如下：

```
// 当其他币分区买入时调用
public int mrbz_QTC(int mrbzid,int scid, BigDecimal price, BigDecimal num,
String... parameter) throws Throwable {
    serviceResource.openTransactions();
    int coinId = 0;
    try {
        if (price.compareTo(new BigDecimal(0)) <= 0 || price == null) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("价格不能小于零, price= " + price);
            }
            throw new ParameterException("价格不能小于零。");
        }
        if (num.compareTo(new BigDecimal(0)) <= 0 || num == null) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("数量不能小于零, num= " + num);
            }
            throw new ParameterException("数量不能小于零。");
        }
    }
    /**
    * 后一个参数包含</br>
    * 第一个交易密码</br>
    * 第二个 apikey</br>
    * 正常只有一个，存在两个时一定是机器人跑的
    * 所以机器人跑的一定要校验交易密码
    */
    if (parameter.length >= 2 || getMmsr().equals(JymmType.MBJY.toString())) {
        if (StringHelper.isEmpty(parameter[0])) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("交易密码不能为空, Jymm= " + parameter[0]);
            }
            throw new ParameterException("交易密码不能为空。");
        }
        if (!Jymm(parameter)) {
```



```

        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("交易密码错误, Jymm= " + parameter);
        }
        throw new ParameterException("交易密码错误。");
    }
}

if (!sfdsd(parameter)) {
    if (LOGGER.isWarnEnabled()) {
        LOGGER.warn("该账号已被锁定, sfdsd= " + parameter);
    }
    throw new ParameterException("该账号已被锁定, 请致电客服询问详情。");
}

int userid = 0;
if (parameter.length >= 2) {
    userid = getUserId(parameter[1]);
} else {
    userid = serviceResource.getSession().getAccountId();
}

if (userid <= 0) {
    if (LOGGER.isWarnEnabled()) {
        LOGGER.warn("用户不存在, userid= " + userid);
    }
    throw new ParameterException("用户不存在, 请联系客服。");
}

BigDecimal mr_min = new BigDecimal(0); // 买入最小交易价
BigDecimal mr_max = new BigDecimal(0); // 买入最大交易价
IsPass sfkqjy = null; // 是否开启交易
IsPass sfxs = null; // 是否显示
BigDecimal mr_rzdcjl = new BigDecimal(0); // 日买入最大成交量
BigDecimal mr_rzdcjl_sc = new BigDecimal(0); // 日买入最大成交量_市场
IsPass zdl_limit = null; // 是否开启成交量限制
try (Connection connection = getConnection(P2PConst.DB_USER)) {
    try (PreparedStatement ps = connection.prepareStatement("SELECT F08,F09,
F12,F13,F21,F23,F25 FROM T6015 WHERE F01=? ")) { // 锁定市场表
        ps.setInt(1, scid);
        try (ResultSet re = ps.executeQuery()) {
            while (re.next()) {
                mr_min = re.getBigDecimal(1);
                mr_max = re.getBigDecimal(2);
                sfkqjy = EnumParser.parse(IsPass.class, re.getString(3));
                sfxs = EnumParser.parse(IsPass.class, re.getString(4));
                mr_rzdcjl = re.getBigDecimal(5);
                mr_rzdcjl_sc = re.getBigDecimal(6);
            }
        }
    }
}

```



## 区块链：交易系统开发指南

```

        zdl_limit = EnumParser.parse(IsPass.class, re.getString(7));
    }
}
}
if (sfkqjy == null || sfkqjy == IsPass.F) {
    if (LOGGER.isWarnEnabled()) {
        LOGGER.warn("交易未开始, sfkqjy= " + sfkqjy);
    }
    throw new ParameterException("交易未开始。");
}
if (sfxs == null || sfxs == IsPass.F) {
    if (LOGGER.isWarnEnabled()) {
        LOGGER.warn("显示状态异常, sfxs= " + sfxs);
    }
    throw new ParameterException("显示状态异常。");
}
if ("S".equals(zdl_limit.name())) {
    // 判断日最大成交量_市场
    BigDecimal dq_mr_rcjl_sc = selectBigDecimal(P2PConst.DB_USER,
        "SELECT SUM(F05) FROM t6017 WHERE F02=? AND to_days(F09) = to_days(now())
AND F08 <> ? AND F07=?", scid, EntrustType.YCX, JyztStatus.MR);
    if (dq_mr_rcjl_sc.add(num).compareTo(mr_rzdcjl_sc) > 0) {
        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("超出市场日买入最大成交量, 市场日买入最大成交量: mr_rzdcjl_sc= "
+ mr_rzdcjl_sc + "市场当日已买入总量: dq_mr_rcjl_sc" + dq_mr_rcjl_sc.add(num));
        }
        throw new ParameterException("市场日买入最大成交量不能大于" + mr_rzdcjl_sc);
    }
    // 非特殊账户时进行校验
    if (!isTszh()) {
        // 判断日最大成交量
        BigDecimal dq_mr_rcjl = selectBigDecimal(P2PConst.DB_USER, "SELECT
SUM(F05) FROM t6017 WHERE F02=? AND F03=? AND to_days(F09) = to_days(now()) AND
F08 <> ? AND F07=?", scid, userid, EntrustType.YCX, JyztStatus.MR);
        if (dq_mr_rcjl.add(num).compareTo(mr_rzdcjl) > 0) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("超出用户日买入最大成交量, 用户日买入最大成交量: mr_rzdcjl= "
+ mr_rzdcjl + "用户当日买入总量: dq_mr_rcjl" + dq_mr_rcjl.add(num));
            }
            throw new ParameterException("用户日买入最大成交量不能大于" + mr_rzdcjl);
        }
    }
}
}

```

```

    }
    // 买入币种简称
    String mr_bzjc = selectString(P2PConst.DB_USER, "SELECT F03 FROM T6013 WHERE
F01=?", mrbzid);
    BigDecimal mr_QTC = num.multiply(price).setScale(8, BigDecimal.ROUND_HALF_
DOWN); // 买入金额
    // 非特殊账户时进行校验
    if (!isTszh()) {
        if (mr_QTC.compareTo(mr_min) <= 0) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("委托金额不能小于"+mr_min+mr_bzjc+", 买入委托金额为
mr_QTC= "+mr_QTC);
            }
            throw new ParameterException("委托金额不能小于"+mr_min+mr_bzjc);
        }
    }
    // 查 T6025 用户虚拟币表
    BigDecimal zh_ky_QTC = selectBigDecimal(P2PConst.DB_USER, "SELECT F04 FROM
T6025 WHERE F02=? AND F03=? FOR UPDATE", userid, mrbzid); // 锁定用户虚拟币资产表
    if (mr_QTC.compareTo(zh_ky_QTC) > 0) {
        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("账户余额不足, 账户可用余额 zh_ky_QTC"+zh_ky_QTC+", 买入委托
金额为 mr_QTC= "+mr_QTC);
        }
        throw new ParameterException("账户余额不足");
    }
    // 修改用户虚拟币表信息-F05 冻结币种/F04 可用币种
    execute(getConnection(P2PConst.DB_USER), "UPDATE T6025 SET F05=F05+?, F04=
F04-? WHERE F02=? AND F03=?", mr_QTC, mr_QTC, userid, mrbzid);
    coinId = insert(getConnection(P2PConst.DB_USER), "INSERT INTO T6017 SET
F02=?, F03=?, F04=?, F05=?, F06=0, F07=?, F08=?, F09=CURRENT_TIMESTAMP()", scid,
userid, price, num, JyztStatus.MR, EntrustType.JYZ);
    if (LOGGER.isInfoEnabled()) {
        LOGGER.info("买入成功:scid=" + scid + ",userid=" + userid + ",price=" +
price + ",num=" + num + ",type=" + JyztStatus.MR.getName());
    }
} catch (Throwable e) {
    serviceResource.rollback();
    e.printStackTrace();
    if (LOGGER.isErrorEnabled()) {
        LOGGER.error(e.getMessage(), e);
    }
    throw new LogicalException(e.getMessage());
}

```

```

    }
    return coinId;
}

```

## 4.1.2 财务中心

### 1. 我的资产

我的资产界面如图 4-5 所示，展示了用户有余额的法币或虚拟币信息。



图 4-5

与法币余额相关的表：ust\_account，其结构如表 4-1 所示。

表 4-1

字段	类型	字段含义
id	int	自增 ID
balance	decimal	余额
freeze	decimal	冻结金额
available	decimal	可用金额
productPoints	decimal	商品积分
consumptionPoints	decimal	消费积分

与虚拟币余额相关的表：coin\_account，其结构如表 4-2 所示。

表 4-2

字段	类型	字段含义
id	int	自增 ID
userId	int	用户 ID
bid	int	虚拟币 ID
available	decimal	可用金额
freeze	decimal	冻结金额



后台根据用户编号查询 `ust_count`，得到用户的 UST 余额信息，根据用户编号币种编号查询得到用户的虚拟币余额信息。两种类型的总余额、冻结金额和可用金额，随着交易、转出的进行会有相应的变化，但总余额一直等于可用金额加上冻结金额。

## 2. 转入资产

转入资产流程图如图 4-6 所示。

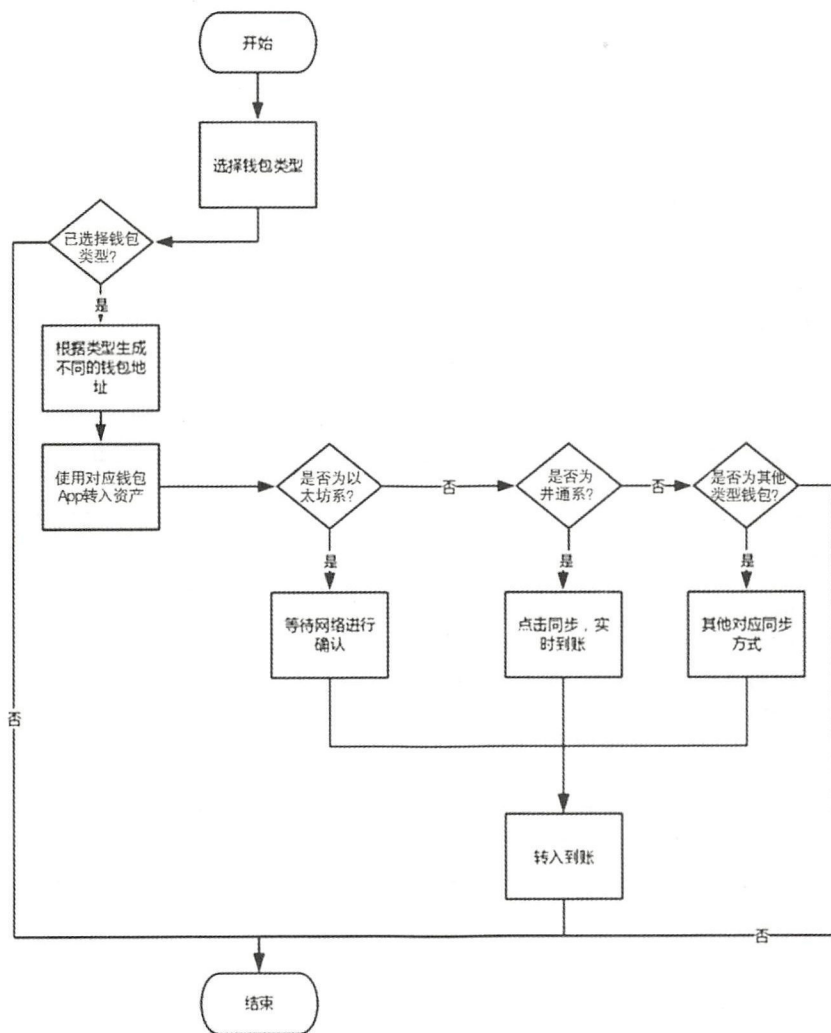


图 4-6



比如生成 SWTC 钱包，SWTC 用户钱包信息表为 wallet\_swt，其结构如表 4-3 所示。

表 4-3

字段	类型	字段含义
id	Int	自增 ID
userId	int	用户 ID
address	varchar	钱包地址
password	varchar	钱包私钥
status	enum	是否失效
createTime	datetime	创建时间
invalid Time	datetime	失效时间

生成钱包地址代码如下：

```
/**
 * @param bid
 * @创建井通钱包地址
 * @throws SQLException
 */
private String swt(int bid, String jc) throws SQLException {
    int accountId = serviceResource.getSession().getAccountId();
    String wallet = "";
    String ip = selectString(P2PConst.DB_USER, "SELECT F18 FROM T6013 WHERE
F01=?", bid);// IP
    if (!StringHelper.isEmpty(jc) && !StringHelper.isEmpty(ip)) {
        String wallet_ = "wallet_" + jc;
        StringBuffer s = new StringBuffer();
        s.append("SELECT F03 FROM ").append(wallet_).append(" AS T6");
        s.append(" WHERE T6.userId=? AND T6.status=?");
        wallet = selectString(P2PConst.DB_USER, s.toString(), accountId,
IsPass.F);
        if (StringHelper.isEmpty(wallet)) {
            StringBuffer url = new StringBuffer(ip).append("/v2/wallet/new");
            String result = HttpRequest.sendGet(url.toString());
            Map<String, Object> reault_map = JsonUtil.GsonToMaps(result);
            if (reault_map != null && reault_map.get("success") != null
                && !StringHelper.isEmpty(reault_map.get("success").toString())
                && Boolean.parseBoolean(reault_map.get("success").toString())) {
                try {
                    Map<String, String> wallet = (Map<String, String>)
reault_map.get ("wallet");
```

```

        String address = wallet.get("address");// 井通钱包地址
        String secret = wallet.get("secret");// 井通钱包私钥
        if (!StringHelper.isEmpty(address) && !StringHelper.isEmpty
(secret)) {
            String jm = MyCrypt.myEncode(secret);
            execute(getConnection(P2PConst.DB_USER),
"INSERT INTO " + wallet_ + " SET F02=?,F03=?,F04=?,F05=?,F06= CURRENT_TIMESTAMP(),
F07=DATE_ADD(NOW(),INTERVAL 1 YEAR)",accountId, address, jm, IsPass.F);
        }
    } catch (Exception e) {
        System.out.printf("创建井通钱包地址失败, 返回结果: %s", result);
    }
}
wallet = selectString(P2PConst.DB_USER, s.toString(), accountId,
IsPass.F);
}
}
return wallet;
}

```

转入 SWTC 资产后同步数据代码如下:

```

public void synchronization(int bid, String address, String currency, String
issuer) throws Throwable {
    serviceResource.openTransactions();
    try {
        int userid = serviceResource.getSession().getAccountId();
        int id= selectInt(P2PConst.DB_USER, "SELECT F01 FROM T6025 WHERE F02=?
AND F03=? ", userid, bid);// 锁定用户虚拟币账户表
        selectInt(P2PConst.DB_USER, "SELECT F01 FROM T6025 WHERE F01=? FOR
UPDATE ", id);
        String ip=selectString(P2PConst.DB_USER, "SELECT F18 FROM T6013 WHERE
F01=?", bid);// IP
        StringBuffer s = new StringBuffer(ip).append("/v2/accounts/").append
(address).append("/payments");
        s.append("?destination_account=").append(address);
        s.append("&direction=").append("incoming");
        s.append("&exclude_failed=").append(true);
        s.append("&results_per_page=").append(50);
        s.append("&page=").append(1);
        if (LOGGER.isInfoEnabled()) {
            LOGGER.info("userid:"+userid+", 查询"+currency+"交易记录的 url="
+s.toString());
        }
    }
}

```



## 区块链：交易系统开发指南

```

        tring result = HttpRequest.sendGet(s.toString());
        Jyjl jyjl = (Jyjl) JSONObject.toBean(JSONObject.fromObject(result),
Jyjl.class);
        if (jyjl.success) {
            Payments[] payments = jyjl.payments;
            if (payments != null) {
                for (Payments payment : payments) {
                    if (!StringHelper.isEmpty(payment.result) && payment.result.
equals("tesSUCCESS") && payment.amount.currency.equals(currency) && "received".
equals(payment.type) && payment.amount.issuer.equals(issuer)) {
                        int t6023_id = selectInt(P2PConst.DB_USER, "SELECT F01 FROM
wallet_3 WHERE F02=?", payment.hash);
                        // 哈希值是否存在
                        if (t6023_id == 0) {
                            String bjc = payment.amount.currency;
                            if (payment.amount.currency.equals("KQC")) {
                                bjc = "HNB";
                            }
                            if (payment.amount.currency.equals("GTA")) {
                                bjc = "MAE";
                            }
                            int wallet_id = selectInt(P2PConst.DB_USER,
"SELECT F01 FROM wallet_" + bjc + " WHERE F02=? AND F03=? AND F05=?", userid,
address, IsPass.F);
                            if (wallet_id > 0) {
                                // 转入的可用金额 ky, 默认等于本次转入总金额
                                BigDecimal ky = new BigDecimal(payment.amount.value);
                                // 判断是不是 SWT 首次充值 (如果该地址在 wallet_3 表中不存在, 则是首次充值)
                                if (payment.amount.currency.equals("SWT")) {
                                    boolean isFirstRecharge4SWT = selectInt(P2PConst.DB_USER,
"SELECT count(1) FROM wallet_3 WHERE F07=?", wallet_id)>0?false:true;
                                    if (isFirstRecharge4SWT) {
                                        // 若是首次充值, 转入的可用金额 ky=本次转入总金额-井通链的固有冻结金额 20
                                        ky = ky.subtract(new BigDecimal(20));
                                    }
                                }
                                // ADD baigang 2018-4-22 SWT 首次转入后实际入账金额由转入金额改为可用金额 (转
                                // 入金额减去冻结金额) --end
                                if (ky.compareTo(new BigDecimal(0)) > 0) {
                                    int wallet_3 = insert(getConnection(P2PConst.DB_USER),
"INSERT INTO wallet_3 SET F02=?, F03=CURRENT_TIMESTAMP(), F04=?, F07=?,
F10=?, F11=?", // 增加 swt 转入记录
payment.hash, ky, wallet_id, bid, payment.amount.value);

```

```

        execute(getConnection(P2PConst.DB_USER), "INSERT INTO T6012_4 SET
F01=?, F02=?", T6012_3id, result); // 增加转入返回记录
        // 查询同步资产前的余额
        BigDecimal ye = selectBigDecimal(P2PConst.DB_USER,
"SELECT F04 + F05 FROM T6025 WHERE F02= ? and F03 = ?", userid, bid);
        if (ye==null) {
            ye = new BigDecimal(0);
        }
        // ADD baigang 2018-4-22 获取同步资产前的余额, 应该在更新余额操作之前--end
        execute(getConnection(P2PConst.DB_USER), "INSERT INTO T6025 SET F02=?,
F03=?, F04=? ON DUPLICATE KEY UPDATE F02=?, F03=?, F04=F04+?", userid, bid, ky,
userid, bid, ky);
        String coin = "交易哈希:" + payment.hash;
        execute(getConnection(P2PConst.DB_USER),
"INSERT INTO T6026 SET F02=?, F03=?, F04=?, F05=CURRENT_TIMESTAMP(), F06=?, F07=?,
F08=?, F09=?, F10=?", userid, bid, XlbType.ZR, ky, T6012_3id, coin, ye,
ye.add(ky));
        execute(getConnection(P2PConst.DB_USER), "INSERT INTO T6027 SET F02=?,
F03=?, F04=?, F05=?, F06=?, F07=CURRENT_TIMESTAMP(), F08=?, F09=?, F10=?",
userid, bid, ky, ky, payment.counterparty, address, IsPass.S, coin);
        //System.out.println("同步资产, 转入金额: " + payment.amount.value + "可用:
" + ky);
        if(LOGGER.isInfoEnabled()) {
            LOGGER.info("userid:"+userid+"-----同步资产, 转入金额: " +
payment.amount.value+"-----币种:"+currency);
        }
        //添加站内信息
        sms(userid, ky, bjc);
    }
}
}
}
}
}
}
} catch (Exception e) {
    serviceResource.rollback();
    if (LOGGER.isErrorEnabled()){
        LOGGER.error(e.getMessage(), e);
    }
    throw new ParameterException(e.getMessage());
}
}
}

```



### 3. 转出资产

转出资产流程图如图 4-7 所示。

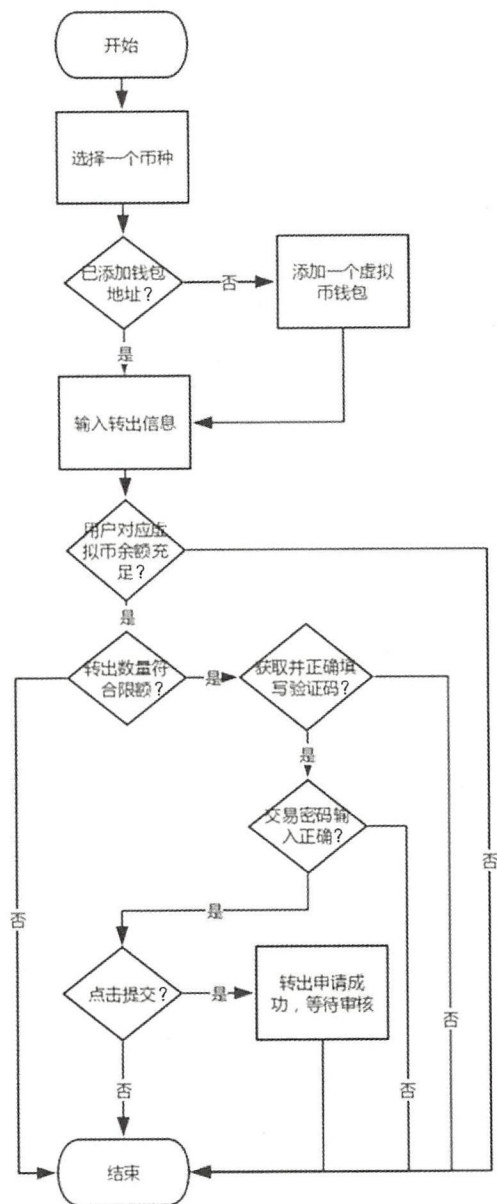


图 4-7

转出资产积分代码如下：

```

/* *
 * @params bid(币种 ID); count(数量); tradeCode(交易密码)
 * walletId(钱包 ID); googleCode(谷歌认证)
 * @转出虚拟币到用户钱包
 * @throws Throwable
 */
public void addXlbZc(int bid, BigDecimal count, String tradeCode, int
walletId, String googleCode) throws Throwable {
    serviceResource.openTransactions();
    try {
        if (bid <= 0) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("币不存在 bid"+bid);
            }
            throw new ParameterException("币不存在。");
        }
        if (count == null || count.compareTo(new BigDecimal(0)) <= 0) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("转出数量异常 count"+count);
            }
            throw new ParameterException("转出数量异常");
        }
        if (walletId <= 0) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("接收地址不能为空 walletId "+ walletId);
            }
            throw new ParameterException("接收地址不能为空。");
        }
        if (StringHelper.isEmpty(tradeCode)) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("请输入提现密码 tradeCode "+ tradeCode);
            }
            throw new ParameterException("请输入提现密码。");
        }
        if (!googleNick(googleCode)) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("请输入提现密码 googleCode"+googleCode);
            }
            throw new ParameterException("请输入谷歌认证");
        }
    }
}

```

## 区块链：交易系统开发指南

```

    }
    if (!Jymm(tradeCode)) {
        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("交易密码错误 tradeCode " + tradeCode);
        }
        throw new ParameterException("交易密码错误。");
    }
    if (!sfds()) {
        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("您的账号异常已被锁定，请联系客服");
        }
        throw new ParameterException("您的账号异常已被锁定，请联系客服。");
    }
    if (!gjsmrz()) {
        throw new ParameterException("请先完成高级实名认证。");
    }
    selectInt(P2PConst.DB_USER, "SELECT F01 FROM T6025 WHERE F02=? FOR UPDATE",
        serviceResource.getSession().getAccountId());
    IsPass is = null; // 是否正常转出
    BigDecimal min_limit = new BigDecimal(0); // 转出最小限制
    BigDecimal max_limit = new BigDecimal(0); // 转出最大限制
    BigDecimal min_fee = new BigDecimal(0); // 转出最低手续费
    BigDecimal max_fee_rate = new BigDecimal(0); // 转出最高手续费率
    String sell = "";
    try (Connection connection = getConnection(P2PConst.DB_USER)) {
        try (PreparedStatement ps = connection
            .prepareStatement("SELECT F07,F08,F09,F10,F11,F03 FROM T6013 WHERE F01=?")) {
            ps.setInt(1, bid);
            try (ResultSet re = ps.executeQuery()) {
                while (re.next()) {
                    is = EnumParser.parse(IsPass.class, re.getString(1));
                    min_limit = re.getBigDecimal(2);
                    max_limit = re.getBigDecimal(3);
                    min_fee = re.getBigDecimal(4);
                    max_fee_rate = re.getBigDecimal(5);
                    sell = re.getString(6);
                }
            }
        }
    }
}

```



```

if (is == null || is == IsPass.F) {
    if (LOGGER.isWarnEnabled()) {
        LOGGER.warn("维护暂时不能转出 is"+is);
    }
    throw new ParameterException("维护暂时不能转出。");
}
if (count.compareTo(min_limit) < 0) {
    if (LOGGER.isWarnEnabled()) {
        LOGGER.warn("转出数量小于最小转出数量: "+count+"小于"+min_limit);
    }
    throw new ParameterException("转出数量小于最小转出数量。");
}
if (getZbDbzg() > 0) {
    max_limit = new BigDecimal(getZbDbzg());
}
if (count.compareTo(max_limit) > 0) {
    if (LOGGER.isWarnEnabled()) {
        LOGGER.warn("转出数量大于最大转出数量: "+count+"大于"+max_limit);
    }
    throw new ParameterException("转出数量大于最大转出数量。");
}
BigDecimal fee = count.multiply(max_fee_rate).setScale(8, BigDecimal.
ROUND_HALF_UP);
if (sxf.compareTo(min_fee) < 0) {
    fee = min_fee;
}
BigDecimal available = selectBigDecimal(P2PConst.DB_USER, "SELECT F04 FROM
T6025 WHERE F02=? AND F03=?",
serviceResource.getSession().getAccountId(), bid); // 可用数量
if ((count).compareTo(available) > 0) {
    if (LOGGER.isWarnEnabled()) {
        LOGGER.warn("转出数量大于可用数量: "+count+"大于"+ available);
    }
    throw new ParameterException("转出数量大于可用数量。");
}
String coin = "数量: " + count + sell + "手续费: " + fee;
execute(getConnection(P2PConst.DB_USER), "UPDATE T6025 SET F04=F04-?,F05=
F05+? WHERE F02=? AND F03=?", count,count, serviceResource.getSession().
getAccountId(), bid);
execute(getConnection(P2PConst.DB_USER), "INSERT INTO T6028 SET F02=?,F03=?,

```



```
F04=?,F05=?,F06=?,F07=CURRENT_TIMESTAMP(),F09=?",
serviceResource.getSession().getAccountId(),bid,walletId,count.subtract(fee),fee,
coin);
    if (LOGGER.isInfoEnabled()) {
        LOGGER.info("转出成功:userid=" + serviceResource.getSession().
getAccountId() + ",bid=" + bid + ",walletId=" + walletId + ",outCount=" +
count.subtract(fee) + ",fee=" + fee + ", coin =" + coin);
    }
} catch (Exception e) {
    serviceResource.rollback();
    e.printStackTrace();
    if (LOGGER.isErrorEnabled()){
        LOGGER.error(e.getMessage(), e);
    }
    throw new ParameterException(e.getMessage());
}
}
```

#### 4. 冷热钱包服务

为保障用户资产的安全，平台提供了冷热钱包服务，当用户有资产转入时，将先转到热钱包，再转到冷钱包。后台服务定时扫描数据库获得资产同步操作记录，调用区块链接口查询获得最新的充值记录，将充值金额安全转移。

这里以 SWTC 币种为例，冷热钱包转入/转出信息表为 T6012\_3，其结构如表 4-4 所示。

表 4-4

字段名称	字段类型	字段含义
id	int	自增 ID
inHash	varchar	转入交易哈希
inTime	timestamp	转入时间
Arrival	decimal	转入实际到账金额
isOut	enum	是否转出
outTime	timestamp	转出时间
coinName	int	参考 T6012_（币简称）.F01
outHash	varchar	转出交易哈希
toHotFee	decimal	从临时钱包转入热钱包手续费

续表

字段名称	字段类型	字段含义
Bid	int	币种 ID（参考 T6013.F01）
inTotal	decimal	转入金额

冷热钱包转入/转出返回信息表为 T6012\_4，其结构如表 4-5 所示。

表 4-5

字段名称	字段类型	字段含义
id	bigint	参考 T6012_3.F01
inMsg	text	转入返回信息
OutMsg	text	转出返回信息

将资产从用户临时钱包转入热钱包部分代码如下：

```
/**
 * @从用户临时钱包转入热钱包
 * @params Lsqbdz
 * @throws throwable
 */
public void swt_rqb(Lsqbdz l) throws Throwable {
    SwtEntity swt=getSwt();
    String hash = "";
    PtqbEntity pt_r=getZhYe(bjc, l.qbdz);
    BigDecimal zhze_r=new BigDecimal(pt_r.balances.get(0).value);// 账户总额
    BigDecimal zhdj_r=new BigDecimal(pt_r.balances.get(0).freezed);// 账户冻结
    if(zhdj_r.compareTo(new BigDecimal(0))<=0){
        zhdj_r=new BigDecimal(20);
    }
    BigDecimal zhky_r=zhze_r.subtract(zhdj_r);// 账户可用金额
    BigDecimal zcje=l.zrje.subtract(new BigDecimal("0.000012"));// 转出金额
    //（可用金额减去手续费）
    int count = selectInt(getConnection(P2PConst.DB_USER),"SELECT COUNT(F07)
FROM T6012_3 WHERE F07=? AND F05=?", l.T6012_id,IsPass.S);// 转入热钱包的次数
    System.out.println("T6012_ID:"+l.T6012_id+"count:"+count+"转出金额:
"+zcje);
    if(zhky_r.compareTo(new BigDecimal(0))>0&&zhky_r.compareTo(zcje)>=0){
        // 转入热钱包
        String result = getZb(l.qbdz,l.sy,zcje,bjc,swt.r_qbdz);
        System.out.println(res+"转入热钱包"+zhky_r+"用户 ID"+l.userid);
    }
}
```

```

        JSONObject jsonObject = JSONObject.fromObject(res);
        boolean success = jsonObject.getBoolean("success");
        if (success) {
            hash = jsonObject.getString("hash");
        }
        // 更新冷热钱包转入/转出信息
        updateWalletMsg(hash, 1);
        // 更新冷热钱包转入/转出返回信息
        updateWalletReturn(res, 1);
    }
}

```

与 SWTC 币种相关的代码定时扫描用户转入记录表,当发现有未转入热钱包的新记录时就执行转移操作,将用户临时钱包的余额转入平台热钱包,并更新转出到热钱包的状态,下次扫描时将不会转出到热钱包。

从用户钱包转出到热钱包的必备条件:用户临时钱包还有余额没有转出;自动转出到热钱包的数额设置必须小于用户钱包余额;自动转出状态设置为“是”。

将资产从用户临时钱包转入热钱包流程图如图 4-8 所示。

从热钱包转入冷钱包相关代码如下:

```

/**
 * @从平台热钱包转入冷钱包
 * @throws throwable
 */
public void swt_lqb() throws Throwable {
    System.out.println("转入冷钱包 1");
    SwtEntity swt=getSwt();
    if(swt.is!=null&&swt.is==IsPass.S&&!StringHelper.isEmpty(
swt.l_qbdz)){// 是否转入冷钱包
        System.out.println("转入冷钱包 2");
        PtqbEntity pt=getZhYe(bjc, swt.r_qbdz);
        BigDecimal zhze=new BigDecimal(pt.balances.get(0).value);
        BigDecimal zhdj=new BigDecimal(pt.balances.get(0).freezed);
        if(zhdj.compareTo(new BigDecimal(0))<=0){
            zhdj=new BigDecimal(20);
        }
        BigDecimal ky=zhze.subtract(zhdj).subtract(new
BigDecimal("0.000012"));
        System.out.println("热钱包可用:"+ky);
    }
}

```



```
// 大于多少转入冷钱包
if (ky.compareTo(new BigDecimal(0))>0&&ky.compareTo(swt.count)>0) {
    // 转入冷钱包
    String l_res = getZb(swt.r_qbdz, swt.r_sy, ky,bjc,swt.l_qbdz);
    System.out.println(l_res+"转入冷钱包"+ky);
}
}
```

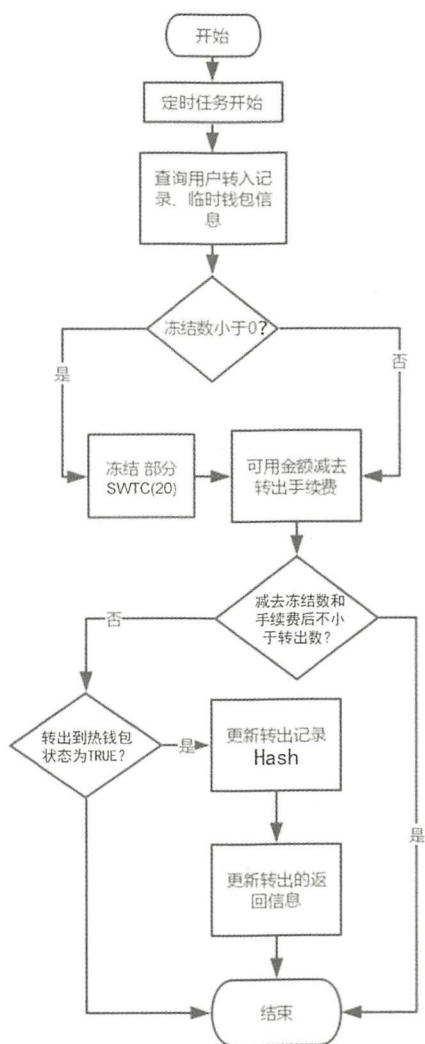


图 4-8

将资产从热钱包转入冷钱包流程图如图 4-9 所示。

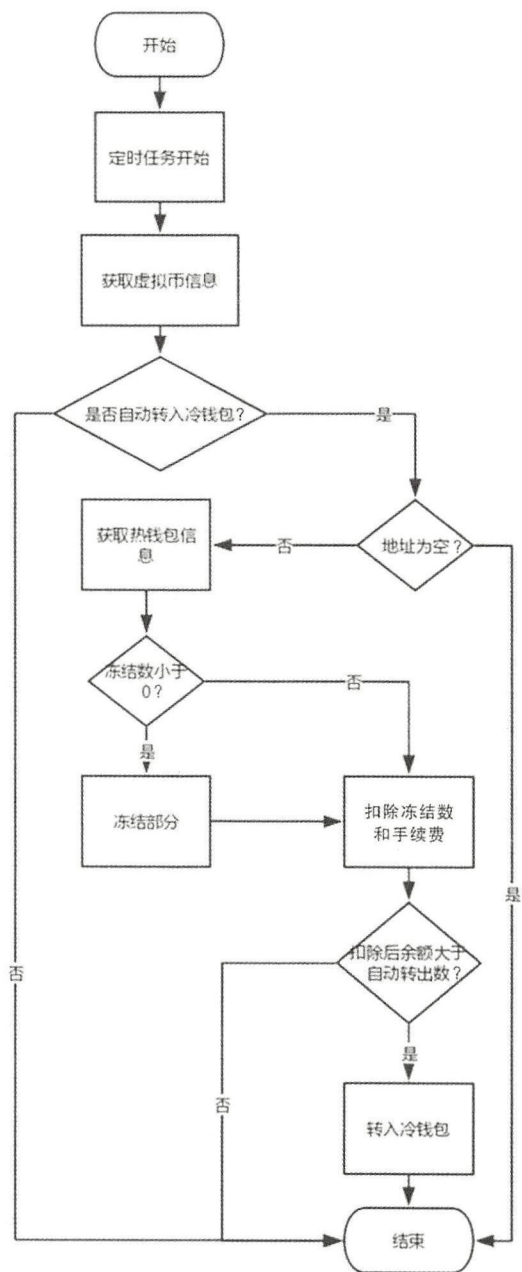


图 4-9



与 SWTC 币种相关的代码定时获取虚拟币热钱包地址, 然后到链上查询, 获取热钱包最新余额信息, 并与平台设置的自动转入值对比, 若符合条件, 则将热钱包可用金额转入冷钱包。若热钱包余额为空, 或者不满足转入条件, 则结束程序, 等到下一次执行定时任务时再进行热钱包转入冷钱包的操作。

## 5. C2C场外交易

买入、卖出界面如图 4-10 所示。

The screenshot displays the 'UST 交易' (UST Trading) interface. It is divided into two main sections: '买入 UST' (Buy UST) on the left and '卖出 UST' (Sell UST) on the right. Each section has a '汇款银行' (Remittance Bank) and '收款银行' (Receiving Bank) dropdown menu, both with a '添加一张银行卡' (Add a bank card) button. Below these are input fields for '买入价 (¥)' (Buy price in ¥) and '卖出价 (¥)' (Sell price in ¥), both set to 6.5. There are also input fields for '买入量 (S)' (Buy quantity in S) and '卖出量 (S)' (Sell quantity in S), both set to 100. At the bottom of each section are '最少买入数量 100' (Minimum buy quantity 100) and '最少卖出数量 100' (Minimum sell quantity 100), along with a '可得 650CNY' (Can get 650CNY) label. Large buttons for '立即买入' (Buy immediately) and '立即卖出' (Sell immediately) are present. Below the trading sections is a table of recent transactions.

用户账号	成交数量	类型	状态	用户账号	成交数量	类型	状态
5****9@qq.com	200.00UST	买入	交易完成	136****7486	372.11UST	卖出	交易完成
187****3736	100.00UST	买入	交易完成	137****5675	198.10UST	卖出	交易完成

图 4-10

C2C 交易表为 T6064, 其结构如表 4-6 所示。

表 4-6

原字段	字段	类型	字段含义
F01	id	Int	自增 ID
F02	userId	Int	用户 ID
F03	coinId	int	币种 ID
F04	createTime	datetime	创建时间
F05	arrival	decimal	实际到账金额
F06	bankId	int	银行卡 ID
F07	fee	decimal	费用





续表

原字段	字段	类型	字段含义
F08	status	int	状态：1，处理中；2，未打款；3，交易完成
F09	processTime	datetime	处理时间
F10	marks	varchar	备注
F11	handlerId	int	处理人 ID
F12	total	decimal	提交数量
F13	rate	decimal	汇率
F14	type	int	类型：1，买入；2，卖出
F15	int	remittanceBankId	汇款银行 ID

C2C 交易买入流程图如图 4-11 所示。

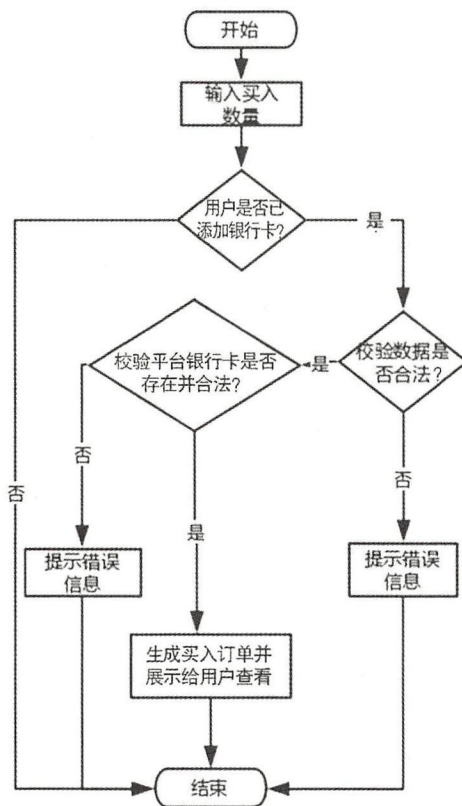


图 4-11



C2C 交易买入代码如下：

```
/**
 *@C2C 交易买入
 *@params amount (数量), bankId (银行卡 ID), bid (币种 ID)
 */
public int c2c_buy(BigDecimal amount, int bankId, int bid) throws Throwable {
    if (amount == null || amount.compareTo(new BigDecimal(0)) <= 0) {
        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("买入数量异常"+"", amount: "+amount");
        }
        throw new ParameterException("买入数量异常");
    }
    if (bankId <= 0) {
        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("请选择银行"+"", bankId: "+ bankId");
        }
        throw new ParameterException("请选择银行。");
    }
    if (bid < 0) {
        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("币不存在,bid="+bid);
        }
        throw new ParameterException("币不存在。");
    }
    ConfigureProvider configureProvider = serviceResource.getResource
        (ConfigureProvider.class);
    BigDecimal min = new BigDecimal(configureProvider.getProperty
        (SystemVariable.USDT_MR_MIN));
    BigDecimal fl = new BigDecimal(configureProvider.getProperty
        (SystemVariable.USDT_MR_FV));
    if (amount.compareTo(min) < 0) {
        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("买入数量小于最小买入数量"+"", amount: "+amount+", 最小买入
min: "+min);
        }
        throw new ParameterException("买入数量小于最小买入数量");
    }
    int bankProxyId = selectInt(P2PConst.DB_CONSOLE, "SELECT F01 FROM t7017
WHERE F05='S' ORDER BY RAND() LIMIT 1");
    if (bankProxyId <= 0) {
```



```

        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("代理银行异常, bankProxyId=" + bankProxyId);
        }
        throw new ParameterException("代理银行异常");
    }
    int accountId = serviceResource.getSession().getAccountId();
    int ddid = insert(getConnection(P2PConst.DB_USER), "INSERT INTO T6064 SET
F02=?, F03=?, F04=CURRENT_TIMESTAMP(), F05=?, F06=?, F07=0, F12=?, F13=?, F14=?, F15=
?", accountId, bid, amount, bankId, amount, fl, C2cType.MR.getType(),
bankProxyId);
    if (LOGGER.isInfoEnabled()) {
        LOGGER.info("c2c_mr [userId = " + accountId + ", amount = " + amount + ",
bankId = " + bankId + ", bid=" + bid + ", result=" + ddid + "]");
    }
    return ddid;
}

```

C2C 交易卖出流程图如图 4-12 所示。

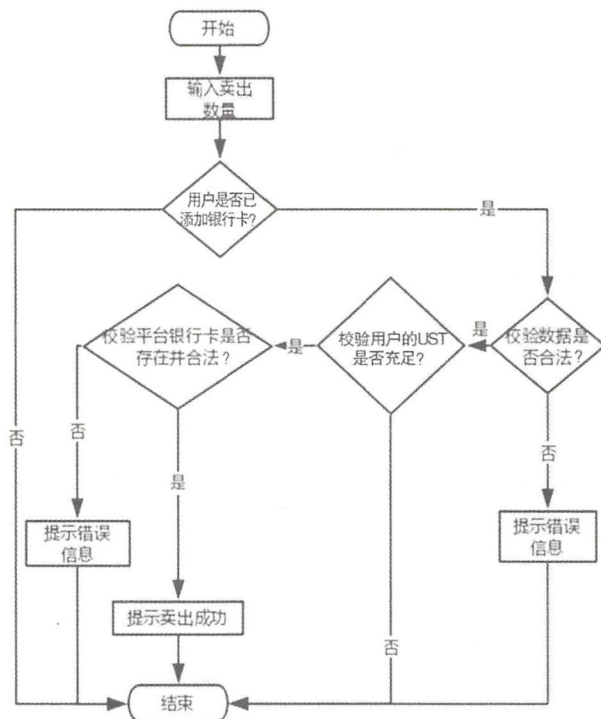


图 4-12





C2C 交易卖出代码如下：

```

/* *
 *@C2C 交易卖出
 *@params amount (数量), bankId (银行卡 ID), bid (币种 ID)
 */
public void c2c_sell(BigDecimal amount, int bankId, int bid) throws Throwable {
    serviceResource.openTransactions();
    try {
        if (amount == null || amount.compareTo(new BigDecimal(0)) <= 0) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("卖出数量异常, amount="+amount);
            }
            throw new ParameterException("卖出数量异常");
        }
        if (bankId <= 0) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("未选择银行, bankId =" + bankId);
            }
            throw new ParameterException("请选择银行。");
        }
        if (bid < 0) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("币不存在, bid="+bid);
            }
            throw new ParameterException("币不存在。");
        }
        ConfigureProvider configureProvider = serviceResource.getResource
        (ConfigureProvider.class);
        BigDecimal min = new BigDecimal(configureProvider.getProperty
        (SystemVariable.USDT_MC_MIN));
        BigDecimal fl = new BigDecimal(configureProvider.getProperty
        (SystemVariable.USDT_MC_FV));
        BigDecimal mcsxfv = new BigDecimal(configureProvider.getProperty
        (SystemVariable.WITHDRAW_FV)); // 卖出手续费率
        if (amount.compareTo(min) < 0) {
            throw new ParameterException("卖出数量小于最小卖出数量");
        }
        int accountId = serviceResource.getSession().getAccountId();
        if (bid > 0) {
            int t6025id = selectInt(P2PConst.DB_USER, "SELECT F01 FROM T6025

```





## 区块链：交易系统开发指南

```

WHERE F02=? AND F03=? ", accountId, bid);// 锁定用户虚拟币账户表
    if (t6025id <= 0) {
        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("币不存在, bid="+t6025id);
        }
        throw new ParameterException("币不存在.");
    }
    BigDecimal zhye = selectBigDecimal(P2PConst.DB_USER, "SELECT F04 FROM
T6025 WHERE F01=? FOR UPDATE", t6025id);
    if (amount.compareTo(zhye) > 0) {
        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("账户数量不足, 账户余额 zhye="+zhye+", amount=
"+amount);
        }
        throw new LogicalException("账户数量不足");
    }
    execute(getConnection(P2PConst.DB_USER), "UPDATE T6025 SET
F04=F04-?, F05=F05+? WHERE F02=? AND F03=?", amount, amount, accountId, bid);
    if (LOGGER.isInfoEnabled()) {
        LOGGER.info("卖出虚拟币成功, bid="+bid+", userid="+accountId+", 数量
amount="+amount);
    }
}
if (bid == 0) {
    BigDecimal zhye = selectBigDecimal(P2PConst.DB_USER, "SELECT F04 FROM
T6023 WHERE F01=? FOR UPDATE", accountId);
    if (amount.compareTo(zhye) > 0) {
        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("账户数量不足, 账户余额 zhye="+zhye+", amount=
"+amount);
        }
        throw new LogicalException("账户余额不足");
    }
}
// 提现成功更新用户账户总表
execute(getConnection(P2PConst.DB_USER), "UPDATE T6023 SET
F03=F03+?, F04= F04-? WHERE F01=?", amount, amount, accountId);
if (LOGGER.isInfoEnabled()) {
    LOGGER.info("卖出 UST 成功, bid="+bid+", userid="+accountId+", 数量
amount="+amount);
}
}

```



```

// 添加 C2C 交易记录
execute(getConnection(P2PConst.DB_USER),"INSERT INTO T6064 SET
F02=?, F03=?, F04=CURRENT_TIMESTAMP(), F05=?, F06=?, F07=?, F12=?, F13=?, F14=?",
accountId, bid, amount.subtract(amount.multiply(mcsxfv)), bankId,
amount.multiply(mcsxfv), amount, fl, C2cType.MC.getType());
if (LOGGER.isInfoEnabled()) {
    LOGGER.info("交易成功
[userid="+accountId+", bid="+bid+", amount="+amount+",
mcsxf="+amount.multiply(mcsxfv)+" , txsl="+amount+" , sjdz="+amount.subtract(amoun
t.multiply(mcsxfv))+", 交易类型 type="+C2cType.MC.getType());
}
} catch (Throwable e) {
    serviceResource.rollback();
    if (LOGGER.isErrorEnabled()){
        LOGGER.error(e.getMessage(), e);
    }
    throw new LogicalException(e.getMessage());
}
}

```

## 6. 交易记录

成交记录表为 trade\_records，其结构如表 4-7 所示。

表 4-7

原字段	字段	类型	字段含义
F01	id	int	自增 ID
F02	markId	int	市场 ID
F03	buyId	int	买方 ID
F04	sellId	int	卖方 ID
F05	price	decimal	成交价格
F06	count	decimal	成交数量
F07	sum	decimal	成交金额
F08	buyFee	decimal	买方手续费
F09	sellFee	decimal	卖方手续费
F10	type	varchar	交易类型：买入、卖出
F11	time	datetime	成交时间



续表

原字段	字段	类型	字段含义
F12	buyEntrustId	int	买方委托 ID
F13	sellEntrustId	int	卖方委托 ID

市场表为 market\_datas，其结构如表 4-8 所示。

表 4-8

原字段	字段	类型	属性含义
F01	id	int	自增 ID
F02	partitionId	int	分区 ID
F03	buyCoinId	int	买方币种 ID
F04	sellCoinId	int	卖方币种 ID
F05	digits	int	小数位数
F06	buyFee	decimal	买入手续费率
F07	sellFee	decimal	卖出手续费率
F08	minBuyPrice	decimal	买入最小交易价
F09	maxBuyPrice	decimal	买入最大交易价
F10	minSellPrice	decimal	卖出最小交易价
F11	maxSellPrice	decimal	卖出最大交易价
F12	status	enum	是否开启交易
F13	isShow	enum	是否显示
F14	time	decimal	交易时间
F15	order	int	排序值
F16	buyMin	decimal	买入最小量
F17	buyMax	decimal	买入最大量
F18	sellMin	decimal	卖出最小量
F19	sellMax	decimal	卖出最大量
F20	isDelete	enum	是否删除
F21	buyMaxDaily	decimal	用户日买入量——最大限制
F22	sellMaxDaily	decimal	用户日卖出量——最大限制
F23	marketBuyMaxDaily	decimal	市场日买入量——最大限制





续表

原字段	字段	类型	属性含义
F24	marketSellMaxDaily	decimal	市场日卖出量——最大限制
F25	isLimitCount	enum	是否启用成交量限制

成交记录查询代码如下：

```
/**
 * @params scid(市场 ID); s(交易状态); paging(分页信息)
 * @成交记录查询
 * @return PagingResult pagingResult
 * @throws Throwable
 */
public PagingResult<JlEntity> searchCjJl(int scid, JyztStatus s, Paging
paging) throws Throwable {
    ArrayList<Object> parameters = new ArrayList<Object>();
    StringBuffer sql = new StringBuffer(
"SELECT T6018.time time,T6018.type type,T6018.price price,T6018.count count,
T6018.sum sum,T6018.buyFee buy_fee,T6018.sellFee sell_fee,");
    sql.append("mr.F03 ywm_c,mr.F04 zwm_c, mc.F03 ywm_r,mc.F04 zwm_r,
T6015.digits ws,T6018.buyId mr_id,T6018.sellId mc_id ");
    sql.append("FROM T6018 LEFT JOIN T6015 ON T6018.F02=T6015.id");
    sql.append("LEFT JOIN T6013 mr ON mr.F01=T6015.buyCoinId ");
    sql.append("LEFT JOIN T6013 mc ON mc.F01=T6015.sellCoinId ");
    sql.append("WHERE 1=1");
    sql.append("AND T6015.id=? AND (T6018.buyId=? OR T6018.sellId =?)");
    if (scid <= 0) {
        scid = getScid();
    }
    final int userid = serviceResource.getSession().getAccountId();
    parameters.add(scid);
    parmeters.add(userid);
    parameters.add(userid);
    if (s != null) {
        if (s == JyztStatus.MRMC) {
            sql.append(" AND T6018.buyId =T6018.sellId");
        } else if (s == JyztStatus.MR) {
            sql.append(" AND T6018.buyId =? ");
            sql.append(" AND T6018.buyId<>T6018.sellId");
            parameters.add(userid);
        } else {
```





```

        sql.append(" AND T6018.sellId =? ");
        sql.append(" AND T6018.buyId <>T6018.sellId ");
        parameters.add(userid);
    }
}

sql.append(" ORDER BY T6018.time DESC");
return selectPaging(getConnection(P2PConst.DB_USER), new ArrayParser
<JlEntity>() {
    ArrayList<JlEntity> list = new ArrayList<JlEntity>();
    @Override
    public JlEntity[] parse(ResultSet re) throws SQLException {
        while (re.next()) {
            JlEntity x = new JlEntity();
            x.time = re.getTimestamp(1);
            x.tradeStatus = EnumParser.parse(JyztStatus.class,
re.getString(2));
            x.price = re.getBigDecimal(3);
            x.count = re.getBigDecimal(4);
            x.sum = re.getBigDecimal(5);
            x.buy_fee = re.getBigDecimal(6);
            x.sell_fee = re.getBigDecimal(7);
            x.ywm_m = re.getString(8);
            x.zwm_m = re.getString(9);
            x.ywm_s = re.getString(10);
            x.zwm_s = re.getString(11);
            x.ws = re.getInt(12);
            x.mr_id = re.getInt(13);
            x.mc_id = re.getInt(14);
            if (x.mr_id == userid && x.mc_id != userid) {
                x.type = "买";
                x.sxf = Formater.formatAmount_(x.buy_fee, x.ws) + x.ywm_s;
                x.ys = "buy";
            } else if (x.mc_id == userid && x.mr_id != userid) {
                x.type = "卖";
                x.sxf = Formater.formatAmount_(x.sell_fee, x.ws) + x.ywm_m;
                x.ys = "sell";
            } else {
                x.type = "买/卖";
                x.sxf = Formater.formatAmount_(x.buy_fee, x.ws) + x.ywm_s + "/"
+ Formater.formatAmount_(x.sell_fee, x.ws) + x.ywm_m;
            }
        }
    }
}

```



```

        list.add(x);
    }
    return list == null ? null : list.toArray(new J1Entity [list.
size()]);
    }
    }, paging, sql.toString(), parameters);
}

```

### 4.1.3 个人中心

#### 1. 我的消息

在个人中心中“我的消息”界面如图 4-13 所示。

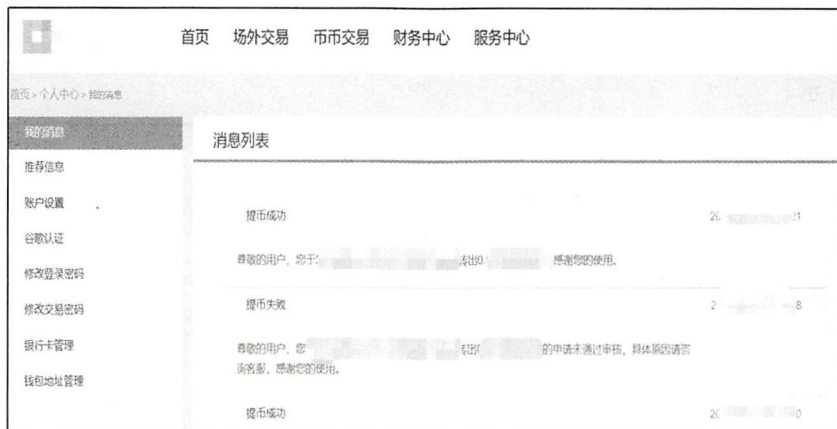


图 4-13

功能说明：

- ◎ 消息功能：显示了与用户操作相关的消息，如 UST 充值、转入资产积分、转出资产积分失败或成功的提示、修改密码的提示和平台赠送活动的提示等消息。
- ◎ 消息模式：当用户有未读消息时，在“个人中心”上会有红点提示，当用户点击查看所有的未读消息后，红点消失。
- ◎ 由于页面读取数据的格式问题，当用户有新消息时不会立刻显示，只有当用户点击其他页面刷新数据时，才会显示消息提示。



与“我的消息”相关的数据库表为 mine\_msg，其结构如表 4-9 所示。

表 4-9

字段	类型	字段含义
F01	int	自增 ID
F02	int	用户编号
F03	varchar	标题
F04	varchar	消息内容
F05	enum	消息状态：'YD'，已读；'WD'，未读
F06	datetime	创建时间

2. 推荐信息

“推荐信息”类似于“我的消息”，但所显示的内容有所不同，“推荐信息”界面如图 4-14 所示。

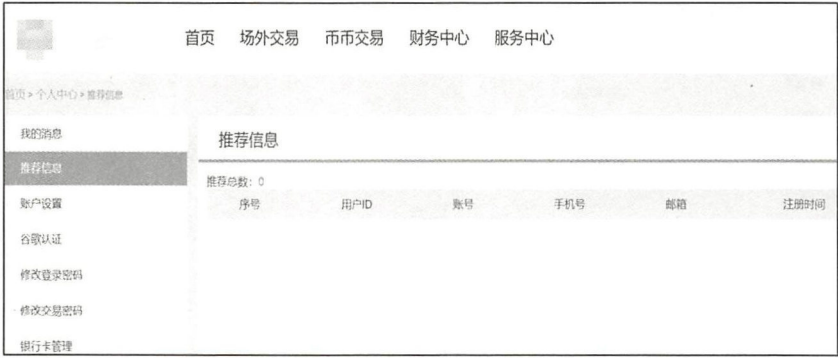


图 4-14

功能说明：

- ◎ 该功能为推荐用户注册所用。
- ◎ 如果推荐人推荐的新用户注册成功，且被推荐的用户在平台有交易时，此处会显示推荐的所有信息，并展示推荐人推荐的新用户。

推荐用户用到的数据库表为 user\_data，其结构如表 4-10 所示。





表 4-10

字段	类型	字段含义
F01	int	用户账号 ID
F02	varchar	姓名
F03	varchar	证件类型
F04	varchar	证件号码
F05	datetime	初级认证时间
F06	varchar	推荐邀请码
F07	varchar	个人邀请码
F08	varchar	国籍
F09	datetime	注册时间
F10	datetime	最后登录时间
F11	varchar	交易密码
F12	varchar	交易密码输入设置
F13	enum	是否通过高级认证; 未提交为 null; 'S', 是; 'F', 否
F14	datetime	高级认证提交时间

- ◎ 被推荐人必须使用“推荐邀请码”注册，使用其他任意字符注册视同未被推荐。
- ◎ 只有当推荐人的“个人邀请码”与被推荐人的“推荐邀请码”都不为空，且两者相同时，在推荐人的“推荐信息”里才会显示被推荐人的信息，从而得到相应的奖励与回报。
- ◎ 被推荐人交易所扣除的手续费和转出资产积分所扣除的手续费，将会由平台和推荐人共同分享。
- ◎ 当推荐人推荐的用户数达到一定数量时，平台可将其推荐的用户的所有手续费收入都奖励给推荐人。
- ◎ 为防止出现“传销式”疯狂推荐的嫌疑，平台设置了推荐级数，就是推荐人只能获得最多两级被推荐人的手续费分成。

查询推荐用户的简单 SQL 语句如下：

```
SELECT COUNT(*)
  FROM user_data
 WHERE F06 =
```



```
(SELECT F07 FROM T6011 WHERE F01=?)
```

此处查询的条件是 id，即推荐人的用户账号 ID。

### 3. 账户设置

账户设置是对用户基本信息和安全验证的设置，主要包括账号信息、邀请码、邀请链接、谷歌认证、登录密码修改和交易密码修改等内容。其中邀请链接与推荐信息有关，新用户通过推荐人发送的带邀请码的邀请链接注册成功后，就会出现在“推荐信息”中。

“账户设置”界面如图 4-15 所示。

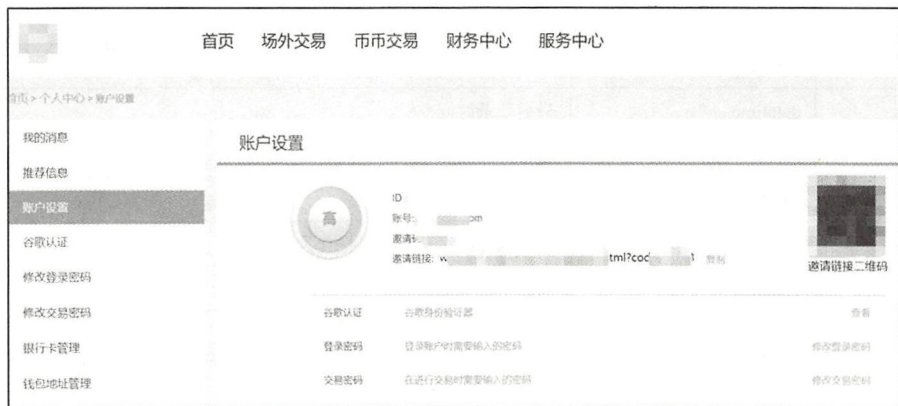


图 4-15

- ◎ 用户可以复制“邀请链接”发送给好友，或者将“邀请链接二维码”截图发送给好友直接扫描注册。
- ◎ 点击“修改登录密码”，将跳转到登录密码的修改页面进行修改操作。
- ◎ 点击“修改交易密码”，将跳转到交易密码的修改页面进行修改操作。

### 4. 谷歌认证

#### (1) 谷歌认证简介

谷歌身份验证器（Google Authenticator）是谷歌公司推出的一款动态口令工具，帮助大家解决账户遭到恶意攻击的问题。

谷歌认证属于第三方认证，当用户在平台进行了谷歌认证后，在转出资产积分时需要





输入手机上动态生成的“认证码”，没有该认证码则无法转出资产积分。谷歌认证为用户的资产安全添加了一层防护。我们强烈建议用户在注册成功后进行谷歌安全认证。

谷歌身份验证器下载地址如下。

iOS 版：<https://itunes.apple.com/cn/app/id388497605?mt=8>

安卓版：<http://m.baidu.com/mip/c/sj.3987.com/mip/down/66726.html>

## (2) 谷歌认证过程

在进行谷歌认证前页面显示如图 4-16 所示。



图 4-16

使用所下载的谷歌认证软件，点击“扫描”，扫描页面中出现的二维码，生成认证码。获取认证码有两种方式，一是通过扫描二维码来获取认证码（推荐使用）；二是通过手工绑定密钥及账户来获取认证码。获取到认证码后应尽快输入到“认证码”输入框中，因为该认证码每 30 秒变化一次。如果未及时将认证码输入到“认证码”输入框中，那么该认证码就会失效过期，需要重新获取认证码。

**注意：**请用户妥善保存好自己的密钥。如果用户没有保存密钥，并且在卸载谷歌认证软件前没有关闭谷歌认证，那么将无法查看、关闭自己的认证信息，谷歌认证软件没有保存历史记录的功能，平台也不负责保存和生成该密钥。

## (3) 谷歌认证成功后的页面

谷歌认证成功后，在谷歌认证页面中会显示“已通过谷歌认证”“查看”“关闭”，







如图 4-17 所示。

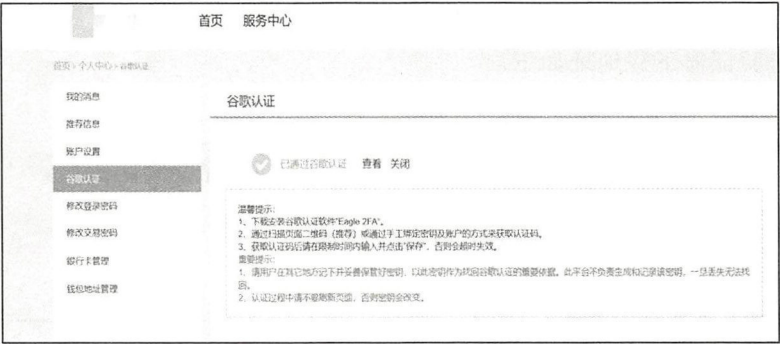


图 4-17

点击“查看”，可以查看认证成功后的 URL 及密钥。此处需要输入谷歌认证软件上的动态认证码，才能进入该页面，如图 4-18 所示。

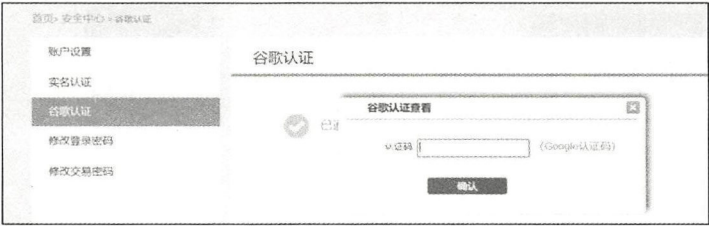


图 4-18

查看页面如图 4-19 所示。

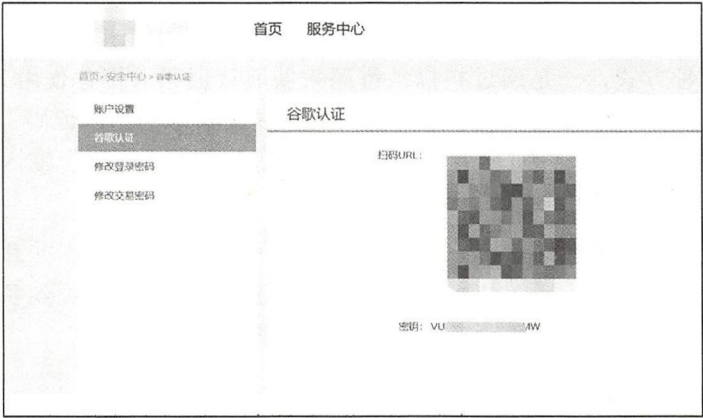


图 4-19





用户在平台开启谷歌认证后，当转出资产时，需要填写扫描本平台二维码生成的账户对应的动态认证码，才能转出成功。

点击“关闭”，可以关闭谷歌认证，如图4-20所示。此处需要输入“登录密码”及谷歌认证软件上的动态认证码。

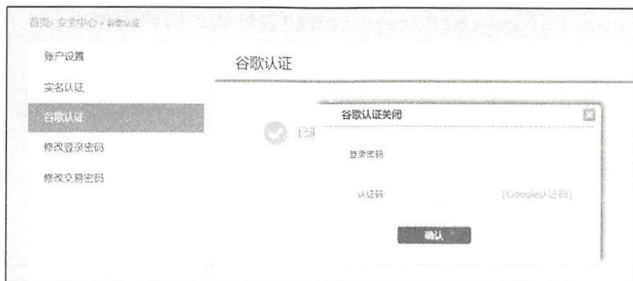


图 4-20

关闭谷歌认证后，手机上之前使用的那个认证码就失效了，可以删除。当再次进行谷歌认证时，需要重新扫描谷歌认证页面上的二维码或手工绑定密钥及账户来获取认证码。

通过谷歌认证的条件是用户下载了谷歌身份验证器，且正确输入了动态认证码。

开通谷歌认证相关代码如下：

```
/**
 * @开通谷歌认证
 * @params secret (私钥), nicknode (谷歌动态认证码), secretUrl (私钥 URL)
 */
public void isgoogleNick(String secret, String nickcode, String syurl) throws
Throwable{
    if(StringHelper.isEmpty(secret)||StringHelper.isEmpty(syurl)){
        throw new ParameterException("密钥错误");
    }
    if(StringHelper.isEmpty(nickcode)){
        throw new ParameterException("认证码不能为空");
    }
    // 输入设备上显示的认证码，编辑并在认证码到期前快速运行它
    long code = Long.valueOf(nickcode) ;
    long t = System.currentTimeMillis();
    GoogleAuthenticator ga = new GoogleAuthenticator();
    ga.setWindowSize(5); // 时间差 30±5 秒
    boolean r = ga.check_code(sy, code, t);
```





```
if(r){
    // 验证码正确
    execute(getConnection(P2PConst.DB_USER), "UPDATE T6036 SET
F02=?,F03=?,F04=? WHERE F01=?", sy,IsPass.S,syurl,
serviceResource.getSession().getAccountId());
}else{
    throw new ParameterException("谷歌验证码错误");
}
}
```

## 5. 修改登录密码

修改登录密码的界面如图 4-21 所示。

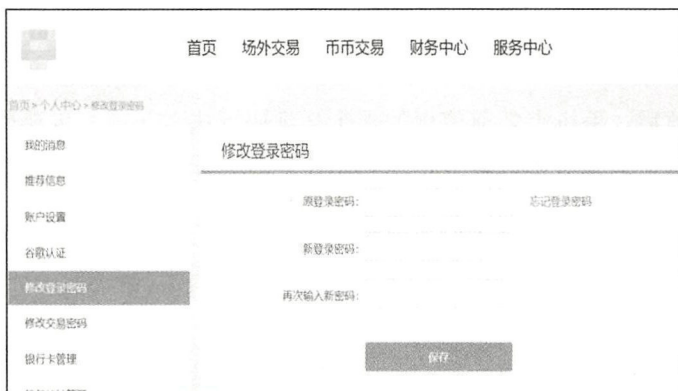


图 4-21

用户可以将安全性低的登录密码修改成安全性较高的密码，但前提是必须输入正确的原登录密码，且两次输入的新登录密码必须一致。

### 注意：

- ⊙ 用户修改密码的 URL 不能随意暴露在页面表单或 Ajax 方法中，以防他人恶意修改账户盗取资产。
- ⊙ 在修改密码时，可以在后台增加必要的判断，比如所输入的密码较为简单时给出相应的提示。
- ⊙ 密码既不能太长，也不能太短。太长的密码容易被遗忘，给自己带来不必要的麻烦；太短的密码又不安全，容易被他人恶意盗号。





- ◎ 后台密码采用 MD5 加密的方式存储，即使数据库被他人非法侵入，其也无法在第一时间获取到用户的真实密码。

修改登录密码的部分代码如下：

```
/* *
 * @修改登录密码的方法
 * @params oldpassword (原密码), newpassword (新密码), repassword (新密码二次输入)
 * @throws Throwable
 */
public void updateLogin(String oldpassword, String newpassword, String
repassword) throws Throwable {
    if (StringHelper.isEmpty(oldpassword)) {
        throw new ParameterException("原密码不能为空。");
    }
    if (StringHelper.isEmpty(newpassword)) {
        throw new ParameterException("新密码不能为空。");
    }
    if (!newpassword.equals(repassword)) {
        throw new ParameterException("两次输入的密码不一致。");
    }
    String _code = getCode(serviceResource.getSession().getAccountId());
    String _tradePsd =
getTradePsd(serviceResource.getSession().getAccountId());
    if (!UnixCrypt.crypt(oldpassword,
DigestUtils.sha256Hex(oldpassword)).equals(_code)) {
        throw new ParameterException("原密码错误。");
    }
    if (!StringHelper.isEmpty(_tradePsd)
&& _tradePsd.equals(UnixCrypt.crypt(newpassword, DigestUtils.sha256Hex
(newpassword)))) {
        throw new ParameterException("不能和交易密码相同。");
    }
    // 判断是否和交易密码相同
    if (!StringHelper.isEmpty(_tradePsd)
&& _tradePsd.equals(MD5.getMd5Value(newpassword))) {
        throw new ParameterException("不能和交易密码相同。");
    }
    // 执行修改登录密码的操作
    upateLoginPsd(newpassword, DigestUtils.sha256Hex(newpassword)),
serviceResource.getSession().getAccountId());
```







```
// 添加修改登录密码成功的站内信息
sms(serviceResource.getSession().getAccountId(), "修改登录密码");
}
```

## 6. 修改交易密码

修改交易密码的界面如图 4-22 所示。

图 4-22

用户可以将安全性低的交易密码修改成安全性较高的密码，但前提是必须输入正确的原交易密码，且两次输入的新交易密码必须一致。

### 注意：

- ① 用户修改交易密码的 URL 不能随意暴露在页面表单或 Ajax 方法中，以防他人恶意进行资产交易和转移。交易密码涉及转出、买入、卖出和提现等关键操作，因此它比登录密码更加重要。
- ② 在修改交易密码时，必须在后台增加必要的判断，如果所输入的密码较为简单，则不能通过验证。
- ③ 密码既不能太长，也不能太短。太长的密码容易被遗忘，给自己带来不必要的麻烦；太短的密码又不安全，容易被他人恶意盗号。
- ④ 后台密码采用 MD5 加密的方式存储，即使数据库被他人非法侵入，其也无法在第一时间获取到用户的真实密码。
- ⑤ 建议定期修改交易密码，最好设置成不规则的密码，防止他人通过穷举法或其他密码破解工具进行篡改，造成不必要的损失。





- ◎ 在与交易相关的地方还是建议开启谷歌认证，即使交易密码被篡改，也能通过谷歌认证这道防线保障资产的安全。

修改交易密码的部分代码如下：

```
/* *
 * @修改交易密码的方法
 * @params oldpassword (原密码), newpassword (新密码), repassword (新密码二次输入)
 * @throws Throwable
 */
public void updateTradePsd(String oldpassword, String newpassword, String
repassword) throws Throwable {
    if (StringHelper.isEmpty(oldpassword)) {
        throw new ParameterException("原密码不能为空。");
    }
    if (StringHelper.isEmpty(newpassword)) {
        throw new ParameterException("新密码不能为空。");
    }
    if (!newpassword.equals(repassword)) {
        throw new ParameterException("两次输入的密码不一致。");
    }
    String _loginPsd = selectString(P2PConst.DB_USER, "SELECT F03 FROM T6010
WHERE F01=?", serviceResource.getSession().getAccountId());
    String _tradePsd = selectString(P2PConst.DB_USER, "SELECT F11 FROM T6011
WHERE F01=?", serviceResource.getSession().getAccountId());
    if (!StringHelper.isEmpty(_loginPsd)
&& _loginPsd.equals(UnixCrypt.crypt(newpassword, DigestUtils.sha256Hex
(newpassword)))) {
        throw new ParameterException("不能和登录密码相同。");
    }
    String md5password = MD5.getMd5Value(oldpassword);
    if (md5password.equals(_tradePsd)) {
        throw new ParameterException("原密码错误。");
    }
    // 执行修改交易密码的操作
    updateTradePsd(UnixCrypt.crypt(newpassword,
DigestUtils.sha256Hex(newpassword)),
```





```
        serviceResource.getSession().getAccountId());
    } else {
        if (!UnixCrypt.crypt(oldpassword,
DigestUtils.sha256Hex(oldpassword)). equals(_tradePsd)) {
            throw new ParameterException("原密码错误。");
        }
        throw new ParameterException("原密码错误。");
    }
    // 执行修改交易密码的操作
    updateTradePsd(UnixCrypt.crypt(newpassword,
DigestUtils.sha256Hex(newpassword)),
        serviceResource.getSession().getAccountId());
    // 添加修改交易密码成功的站内信息
    sms(serviceResource.getSession().getAccountId(), "修改交易密码");
}
}
```

## 7. 银行卡管理

在平台添加的银行卡用来进行场外交易，为了方便快捷地买入、卖出 UST，用户可以通过“银行卡管理”功能对自己的银行卡进行管理，包括查看列表、添加新银行卡和删除银行卡功能。

“银行卡管理”界面如图 4-23 所示。

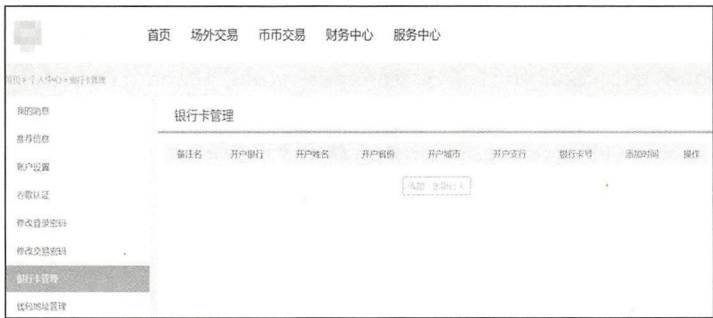


图 4-23

功能说明：

- 在添加银行卡时，需要输入正确的备注名、开户银行、开户省市、开户支行、开





户姓名、银行卡号和交易密码等。

- 目前只做了银行卡格式校验，提示用户输入正确的银行名称、开户银行所在地、开户姓名和银行卡号。若输入错误，则在提交场外交易的申请时将失败。

“添加银行卡”界面如图 4-24 所示。

图 4-24

- 对备注名填写格式未校验，但不能为空。
- 开户银行、开户省市、开户支行、开户姓名、银行卡号和交易密码都不能为空。
- 对银行卡号格式有检验，必须符合银行的特殊格式要求（包含位数等校验规则）方能通过验证，添加成功。
- 对开户银行、开户省市、开户支行、开户姓名的填写，应尽量与所用的银行卡信息一致，以便可以顺利买入/卖出 UST。

用户输入所有必填项后，才能添加银行卡成功。

添加银行卡的流程图如图 4-25 所示。





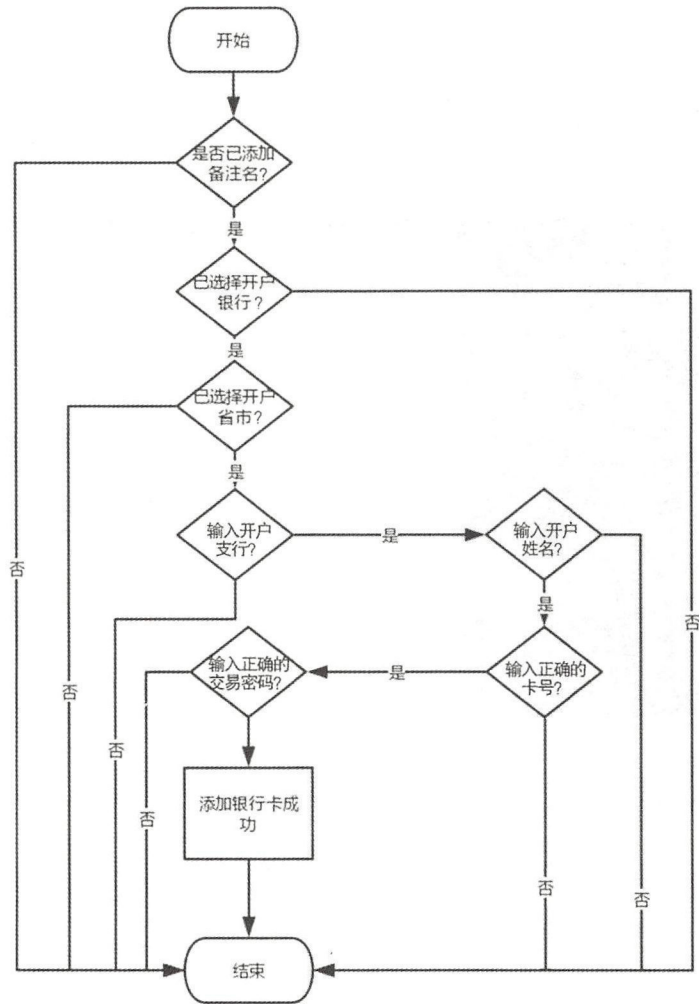


图 4-25

添加银行卡的部分代码如下：

```
/* *
 * @添加银行卡的方法
 * @params label (银行卡标签), bankId (银行 ID), province (开户省份), city (开户城市)
 * subBranche (开户支行), name (开户姓名), bankNum (银行卡号), tradePsd (交易密码)
 * @throws Throwable
 */
public void addBank(String label, int bankId, String province, String city,
```





```
String subBranche, String name, String bankNum, String tradePsd) throws
Throwable {
    if (bankId <= 0) {
        throw new ParameterException("请选择银行。");
    }
    if (StringHelper.isEmpty(province)) {
        throw new ParameterException("开户省份不能为空。");
    }
    if (StringHelper.isEmpty(city)) {
        throw new ParameterException("开户城市不能为空。");
    }
    if (StringHelper.isEmpty(subBranche)) {
        throw new ParameterException("开户支行不能为空。");
    }
    if (StringHelper.isEmpty(name)) {
        throw new ParameterException("开户姓名不能为空。");
    }
    if (StringHelper.isEmpty(bankNum)) {
        throw new ParameterException("银行卡号不能为空。");
    }
    if (!Jymm(tradePsd)) {
        throw new ParameterException("交易密码错误。");
    }
    if (!BankUtil.checkBankCard(bankNum)) {
        throw new ParameterException("输入的银行卡号错误");
    }
    // 查询相同银行卡号的个数
    int i = getBankCount(serviceResource.getSession().getAccountId(), kh);
    if (i > 0) {
        throw new ParameterException("输入的银行卡号已存在");
    }
    // 获取已经存在的银行卡号
    int id = getBankId(serviceResource.getSession().getAccountId(),
bankNum );
    if (id > 0) {
        updateBank(bankId, province, subBranche, BankCardStatus.QY, city,
label, name, id);
    } else {
        addBank(serviceResource.getSession().getAccountId(), bankId, province,
subBranche, bankNum, city, label ,name);
    }
}
```





## 8. 钱包地址管理

钱包作为交易平台关键的一个功能，需要单独管理，可以对钱包地址进行查看、修改、删除等操作。

“钱包地址管理”界面如图 4-26 所示。

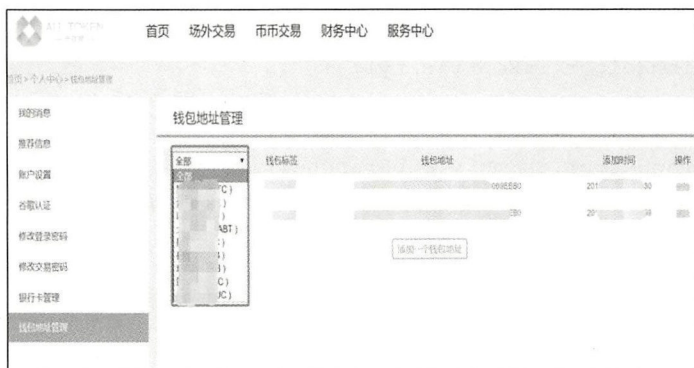


图 4-26

用户在右侧的下拉列表框中选择相应的币种，即可查看所添加的币种对应的钱包地址，目前只支持查看、添加、删除钱包地址的功能。

查看钱包地址的部分代码如下：

```
/* *
 * @ 查看虚拟币钱包地址
 * @ params bid (币种 ID)
 */
public QbdzEntity[] qbdzList(int bid) throws Throwable {
    if(bid<=0){
        return;
    }
    //根据 userid 和 bid 查看用户所选的币种下所有的钱包地址
    QbdzEntity[] re=getWalletAdds(bid,serviceResource.getSession().getAccountId());
    ArrayList<QbdzEntity> list = new ArrayList<QbdzEntity>();
    while (re.next()) {
        QbdzEntity b = new QbdzEntity();
        b.id = re.getInt(1);
        b.enName= re.getString(2);
        b.cnName = re.getString(3);
        b.label= re.getString(4);
        b.addres = re.getString(5);
    }
}
```







```
b.time = re.getTimestamp(6);  
b.coinLogo = re.getString(7);  
list.add(b);  
}  
list == null || list.size() == 0 ? null : list.toArray(new QbdzEntity  
[list.size()]);  
}
```

如图 4-27 所示是添加一个钱包地址的界面，用户通过在下拉列表框中选择相应的币种来添加钱包地址。

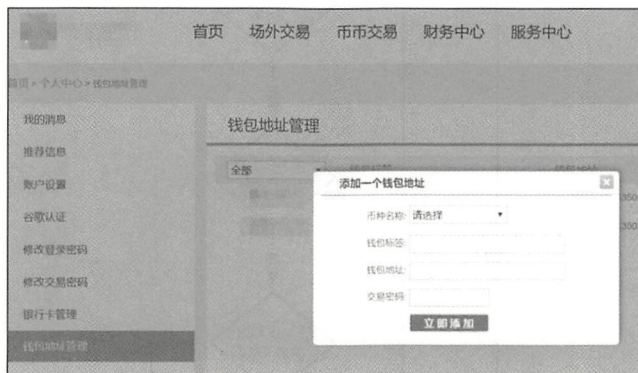


图 4-27

添加钱包地址的流程图如图 4-28 所示。

添加钱包地址须知：

- ◎ 要添加一个钱包地址，需要选择一个币种，否则无法添加。
- ◎ 钱包地址必须与所选择的币种对应，平台只能检测出地址格式规范与否，无法验证是否与所选择的币种对应。由于添加钱包地址所导致的用户损失，平台也无法找回，因此在添加钱包地址时需仔细操作，在转出资产时务必再三确认所选择的地址与转出的币种相对应。
- ◎ 添加钱包地址后暂时无法修改，以防止他人修改钱包地址导致账户不安全。若要修改钱包地址，则需先删除，然后重新添加。





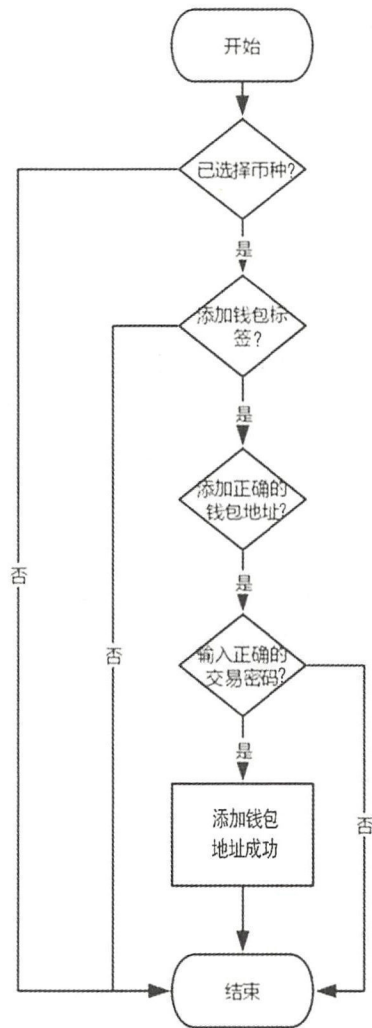


图 4-28

添加钱包地址的部分代码如下：

```
/* *  
 * @ 添加虚拟币钱包地址  
 * @ params label (钱包标签), bid (币种 ID), walletAdds (钱包地址), tradePsd (交易密码)  
 */  
public void addQbdz(String label, int bid, String walletAdds, String tradePsd)  
throws Throwable {
```





```
if (bid < 0) {  
    throw new ParameterException("请选择虚拟币。");  
}  
if (StringHelper.isEmpty(walletAdds)) {  
    throw new ParameterException("钱包地址不能为空。");  
}  
if (!Jymm(tradePsd)) {  
    throw new ParameterException("交易密码错误。");  
}  
// 获得已存在的钱包地址对应的钱包 ID  
int id = getWalletId(serviceResource.getSession().getAccountId(), bid,  
walletAdds, IsPass.F);  
if (id > 0) {  
    updateWalleteAdds (IsPass.S, label, id);  
} else {  
    // 执行添加钱包地址的操作  
    addWalleteAdds (serviceResource.getSession().getAccountId(),  
        bid, label, walletAdds);  
}  
}
```

#### 4.1.4 服务中心

##### 1. 常见问题

“常见问题”界面如图 4-29 所示，主要介绍与平台相关的问题。



图 4-29





这些问题都以文本的形式存储在数据库中，根据类型的不同区分为账户问题、充值提现、转入转出和关于交易等。

常见问题相关表为 questions，其结构如表 4-11 所示。

表 4-11

字段	类型	字段含义
id	int	自增 ID
type	varchar	问题类型
title	varchar	问题标题
createTime	datetime	创建时间
author	int	创建者
views	int	浏览次数
order	int	排序
language	varchar	语言

常见问题正文表为 question\_content，其结构如表 4-12 所示。

表 4-12

字段	类型	字段含义
id	int	问题 ID
content	text	问题内容

其中，type 用来区分问题类型，title 用来显示问题标题，内容保存在 question\_content 中，用 id 关联 questions id。

## 2. 下载中心

下载中心表为 down\_content，其结构如表 4-13 所示。

表 4-13

字段	类型	字段含义
id	int	自增 ID
type	varchar	类型
order	int	排序





续表

字段	类型	字段含义
isPublish	enum	是否发布
title	varchar	标题
source	varchar	来源
summary	varchar	简介
userId	int	用户 ID
addTime	datetime	添加时间
publishTime	datetime	发布时间
time	timestamp	创建时间
isRecommend	enum	是否推荐
order	int	推荐排序
language	varchar	语言

添加下载内容的代码如下：

```
/* *
 * @params ArticleType、Article
 * @添加下载内容
 * @throws Throwable
 */
public int add(ArticleType articleType, Article article) throws Throwable {
    if (articleType == null) {
        throw new ParameterException("没有指定文章类型");
    }
    String title;
    String source;
    String content = null;
    IsPass publishStatus = null;
    IsPass isPass = article.getIstj();
    {
        // 校验
        title = article.getTitle();
        if (StringHelper.isEmpty(title)) {
            throw new ParameterException("标题不能为空。");
        }
        if (title.length() > 60) {
            throw new ParameterException("标题不能超过 60 个字符。");
        }
    }
}
```







```

    }
    source = article.getSource();
    if (!StringHelper.isEmpty(source) && source.length() > 50) {
        throw new ParameterException("文章来源不能超过 50 个字符。");
    }
    publishStatus = article.getPublishStatus();
    if (publishStatus == null) {
        publishStatus = IsPass.F;
    }
    content = article.getContent();
    if (StringHelper.isEmpty(content)) {
        throw new ParameterException("文章内容不能为空");
    }
    if (isPass == null) {
        isPass = IsPass.F;
    }
}

serviceResource.openTransactions();
try (Connection connection = getConnection()) {
    Timestamp now = new Timestamp(System.currentTimeMillis());
    int id = insert(connection, "INSERT INTO T5011 SET type = ?,
order = ?,isPublish = ?,title = ?,source = ?,summary = ?,userId = ?,
addTime = ?,publishTime = ?,isRecommend=?,order=?,language=?",
articleType.name(), article.getSortIndex(), publishStatus.name(), title,
source, article.getSummary(), serviceResource.getSession().getAccountId(),
now, article.publishTime(), isPass, article.getIsorder(), article.getYy());
    execute(connection, "INSERT INTO T5011_1 SET id = ?, content= ?", id,
content);
    return id;
}
}

```

上述代码实现了在后台添加下载内容的功能。管理员登录后台添加合适的下载内容与链接，前台用户在“下载中心”中将会看到对应的下载内容，可以下载使用。

## 4.2 后台管理概述

上一节介绍的是区块链网站的前台功能，本节将具体介绍区块链网站的后台管理系





统。后台管理系统主要用于管理网站前台信息，比如管理文字、图片、影音，以及日常使用文件的发布、更新、删除等操作，同时也包括对会员信息、订单信息、访客信息、平台资金的统计和管理。简单来说，就是对网站数据库和文件的快速操作和管理，以使得前台内容能够得到及时更新和调整。

首先进入后台管理系统首页，如图 4-30 所示。该页面主要统计的是整个平台的注册用户数、用户总资金数及平台总资金的变化情况。在该页面中还可以修改登录密码，以及安全退出。

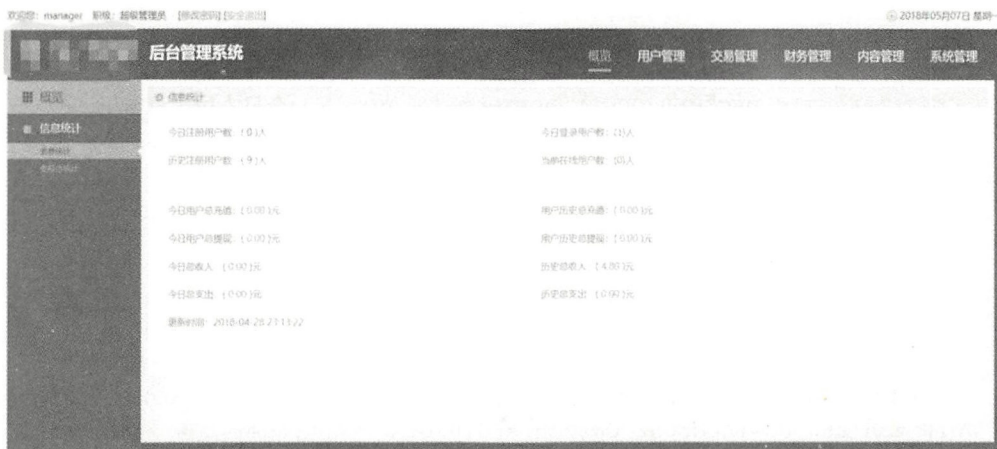


图 4-30

该页面的数据更新是通过数据库的一个事件，每隔 1 小时调用一次更新数据的存储过程来实现的，如图 4-31 所示。



图 4-31





首页信息统计的存储过程如下：

```
BEGIN

    DECLARE _id INT unsigned DEFAULT '0';-- 今日注册用户数
    DECLARE _F02 INT unsigned DEFAULT '0';-- 历史注册用户数
    DECLARE _F03 INT unsigned DEFAULT '0';-- 今日登录用户数
    DECLARE _F04 DECIMAL(20,2) DEFAULT '0.00'; -- 今日用户总充值
    DECLARE _F05 DECIMAL(20,2) DEFAULT '0.00'; -- 用户历史总充值
    DECLARE _F06 DECIMAL(20,2) DEFAULT '0.00'; -- 今日用户总提现
    DECLARE _F07 DECIMAL(20,2) DEFAULT '0.00'; -- 用户历史总提现
    DECLARE _F08 DECIMAL(20,2) DEFAULT '0.00'; -- 人民币总支出
    DECLARE _F09 DECIMAL(20,2) DEFAULT '0.00'; -- 人民币总收入
    DECLARE _F10 DECIMAL(20,2) DEFAULT '0.00'; -- 今日人民币总支出
    DECLARE _F11 DECIMAL(20,2) DEFAULT '0.00'; -- 今日人民币总收入

    SELECT COUNT(*) INTO _id FROM zch_s60.UserInfo WHERE TO_DAYS
(zch_s60.UserInfo.regDate) = TO_DAYS(NOW());-- 今日注册用户数
    SELECT COUNT(*) INTO _F02 FROM zch_s60.UserInfo WHERE TO_DAYS
(zch_s60.UserInfo.regDate) <= TO_DAYS(NOW());-- 历史注册用户数
    SELECT COUNT(*) INTO _F03 FROM zch_s60.UserInfo WHERE TO_DAYS
(zch_s60.UserInfo.lastLoginTime) = TO_DAYS(NOW());-- 今日登录用户数
    SELECT SUM(zch_s60.UserUSTCharge.money) INTO _F04 FROM zch_s60.UserUSTCharge
LEFT JOIN zch_s60.UserAccount ON zch_s60.UserAccount.userId = zch_s60.
UserUSTCharge.userId WHERE zch_s60.UserAccount.is_specialAccount = 'F' AND
zch_s60.UserUSTCharge.status = 'FundsAccount' AND (TO_DAYS(zch_s60.UserUSTCharge.
paymentSuccessTime)=TO_DAYS(NOW()) OR TO_DAYS(zch_s60.UserUSTCharge.createtime)
= TO_DAYS(NOW()));-- 今日用户总充值

    SELECT SUM(zch_s60.UserUSTCharge.money) INTO _F05 FROM zch_s60.UserUSTCharge
LEFT JOIN zch_s60.UserAccount ON zch_s60.UserAccount.userId = zch_s60.
UserUSTCharge.userId WHERE zch_s60.UserAccount.is_specialAccount = 'F' AND
zch_s60.UserUSTCharge.status='FundsAccount' AND (TO_DAYS(zch_s60.UserUSTCharge.
paymentSuccessTime) <= TO_DAYS(NOW()) OR TO_DAYS(zch_s60.UserUSTCharge.
createtime) <= TO_DAYS(NOW()));-- 用户历史总充值

    SELECT SUM(zch_s60.UserUSTWithdrawal.money) INTO _F06 FROM zch_s60.
UserUSTWithdrawal LEFT JOIN zch_s60.UserAccount ON zch_s60.UserAccount.userId
= zch_s60.UserUSTWithdrawal.userId WHERE zch_s60.UserAccount.is_specialAccount =
'F' AND TO_DAYS(zch_s60.UserUSTWithdrawal.loanTime) = TO_DAYS(NOW());-- 今日用
户总提现

    SELECT SUM(zch_s60.UserUSTWithdrawal.money) INTO _F07 FROM zch_s60.
```







```
UserUSTWithdrawal LEFT JOIN zch_s60.UserAccount ON zch_s60.UserAccount.userId
= zch_s60.UserUSTWithdrawal.userId WHERE zch_s60.UserAccount.is_specialAccount =
'F' AND TO_DAYS(zch_s60.UserUSTWithdrawal.loanTime) <= TO_DAYS(NOW());-- 用户
历史总提现
```

```
SELECT SUM(_bPlatformTransactionRecord.inCome) INTO _F08 FROM
_bPlatformTransactionRecord LEFT JOIN zch_s60.UserAccount ON zch_s60.
UserAccount.userId = _bPlatformTransactionRecord.userId WHERE zch_s60.UserAccount.
is_specialAccount = 'F' AND TO_DAYS(_bPlatformTransactionRecord.dateTime) <=
TO_DAYS(NOW());-- 人民币总支出
```

```
SELECT SUM(_bPlatformTransactionRecord.payOut) INTO _F09 FROM
_bPlatformTransactionRecord LEFT JOIN zch_s60.UserAccount ON zch_s60.UserAccount.
userId = _bPlatformTransactionRecord.userId WHERE zch_s60.UserAccount.is_
specialAccount = 'F' AND TO_DAYS(_bPlatformTransactionRecord.dateTime) <=
TO_DAYS(NOW());-- 人民币总收入
```

```
SELECT SUM(_bPlatformTransactionRecord.income) INTO _F10 FROM
_bPlatformTransactionRecord LEFT JOIN zch_s60.UserAccount ON zch_s60.UserAccount.
userId = _bPlatformTransactionRecord.userId WHERE zch_s60.UserAccount.is_specialAccount=
'F' AND TO_DAYS(_bPlatformTransactionRecord.dateTime) = TO_DAYS(NOW());-- 今日
人民币总支出
```

```
SELECT SUM(_bPlatformTransactionRecord.payout) INTO _F11 FROM
_bPlatformTransactionRecord LEFT JOIN zch_s60.UserAccount ON zch_s60.UserAccount.
userId = _bPlatformTransactionRecord.userId WHERE zch_s60.UserAccount.is_
specialAccount='F' AND TO_DAYS (_bPlatformTransactionRecord.dateTime)=TO_DAYS
(NOW());-- 今日人民币总收入
```

```
UPDATE zch_s70.Statistics SET id=_id,F02=_F02,F03=_F03,F04=_F04,F05
=_F05,F06=_F06,F07=_F07,F08=_F08,F09=_F09,F10=_F10,F11=_F11;
```

```
END
```

通过事件对函数每小时一次的调用，可以实时更新该页面数据，让后台操作人员对当日用户注册数及访问量、平台金额及用户资金的变化情况有一个总体的了解。

## 4.2.1 用户管理

### 1. 用户列表

如果想进一步了解当日注册用户数及历史注册用户数的详细信息，则可以点击首页信







息统计当中的今日注册用户数或历史注册用户数，还可以通过点击“用户管理”跳转到“用户列表”页面，如图 4-32 所示。

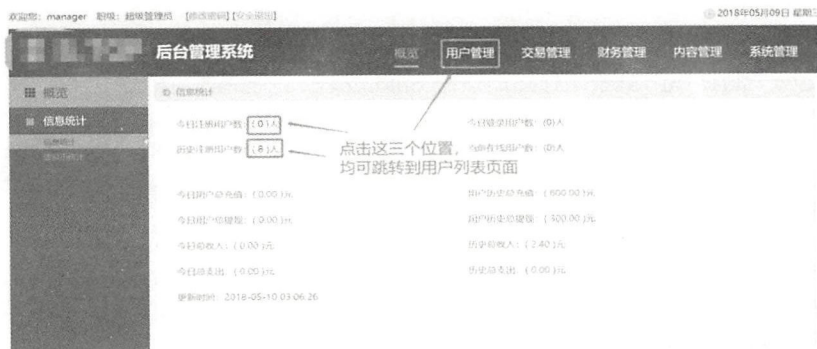


图 4-32

“用户列表”页面为用户管理的默认页面，其显示的内容主要是在该平台注册的用户信息，包括用户 ID、账号、推荐邀请码、个人邀请码、注册时间等。该页面支持对用户数据进行搜索、查看、编辑及锁定等操作，如图 4-33 所示。



图 4-33

### (1) 搜索用户

搜索功能如图 4-33 所示，支持按用户 ID、手机/邮箱（即用户账号）、推荐邀请码、个人邀请码、注册时间段及用户账号是否锁定进行搜索。

条件查询用户列表代码如下：

```
@Right(id = "C_USER_LIST", isMenu = true, name = "用户列表")
public class UserList extends AbstractUserServlet {
```





```

private static final long serialVersionUID = 1L;
@Override
protected void processGet(HttpServletRequest request,
    HttpServletResponse response, ServiceSession serviceSession) throws Throwable
{
    processPost(request, response, serviceSession);
}
@Override
protected void processPost(final HttpServletRequest request, final
    HttpServletResponse response, final ServiceSession serviceSession) throws
    Throwable {
    UserManager userManager = serviceSession.getService(UserManager.class);
    PagingResult<User> users = userManager.userSearch(new UserQuery() {
        @Override
        public int getUserId() {
            return IntegerParser.parse(request.getParameter("userid"));
        }
        @Override
        public String getName() {
            return request.getParameter("name");
        }
        ..... // 省略部分查询代码
        @Override
        public String getUserCode() {
            return request.getParameter("userCode");
        }
    }, new Paging() {
        @Override
        public int getSize() {
            return 10;
        }
        @Override
        public int getCurrentPage() {
            return IntegerParser.parse(request.getParameter(PAGING_CURRENT));
        }
    });
    int count=userManager.getCount();
    request.setAttribute("count", count);
    request.setAttribute("users", users);
    forwardView(request, response, getClass());
}
}

```



首页信息数据库表结构如下：

```
CREATE TABLE `HomeInfo` (  
  `today_register_nums` int(10) unsigned DEFAULT '0' COMMENT '今日注册用户数',  
  `history_register_nums` int(10) unsigned DEFAULT '0' COMMENT '历史注册用户数',  
  `today_login_nums` int(10) unsigned DEFAULT '0' COMMENT '今日登录用户数',  
  `today_recharge_total_amount` decimal(20,8) DEFAULT '0.00000000' COMMENT  
'今日用户总充值',  
  `history_recharge_total_amount` decimal(20,8) DEFAULT '0.00000000'  
COMMENT '用户历史总充值',  
  `today_withdraw_amount` decimal(20,8) DEFAULT '0.00000000' COMMENT '今日  
用户总提现',  
  `history_withdraw_amount` decimal(20,8) DEFAULT '0.00000000' COMMENT '  
用户历史总提现',  
  `CNY_disbursement_amount` decimal(20,8) DEFAULT '0.00000000' COMMENT '  
人民币总支出',  
  `CNY_income_amount` decimal(20,8) DEFAULT '0.00000000' COMMENT '人民币总收入',  
  `today_CNY_disbursement` decimal(20,8) DEFAULT '0.00000000' COMMENT '今  
日人民币总支出',  
  `today_CNY_income` decimal(20,8) DEFAULT '0.00000000' COMMENT '今日人民币  
总收入',  
  `updateTime` timestamp NULL DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP  
COMMENT '更新时间'  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='首页统计表(HomeInfo)';
```

## (2) 查看用户详细情况

若想查看用户的具体信息，点击该用户信息后面的“查看”即可进入用户详细情况页面，该页面显示了用户的账号信息及状态、资产信息、充值记录和提现记录，如图 4-34 所示。



图 4-34





在“用户详细情况”板块中，可以查看用户账号表及用户法币信息表；在“资产信息”板块中，可以查看用户的虚拟币信息表；在“充值记录”板块中，可以查看用户的法币充值记录表；在“提现记录”板块中，可以查看用户的法币提现信息表。

查看用户列表代码如下：

```
@Right(id = "C_USER_LIST_CK", name = "用户列表-查看")
public class ViewUser extends AbstractUserServlet {
    private static final long serialVersionUID = 1L;
    @Override
    protected void processGet(HttpServletRequest request,
        HttpServletResponse response, ServiceSession serviceSession)
        throws Throwable {
        processPost(request, response, serviceSession);
    }
    @Override
    protected void processPost(final HttpServletRequest request,
        HttpServletResponse response, ServiceSession serviceSession)
        throws Throwable {
        UserManager userManager = serviceSession.getService(UserManage.class);
        int userid = IntegerParser.parse(request.getParameter("userid"));
        User user=userManage.get(userid);
        ZhxxEntity rmbZhxx=userManage.rmbZhxx(userid);
        PagingResult<ZhxxEntity> xlbZhxx=userManage.xlbZhxx(userid,new Paging() {
            @Override
            public int getSize() {
                return 10;
            }
            @Override
            public int getCurrentPage() {
                return IntegerParser.parse(request.getParameter(PAGING_CURRENT));
            }
        });
        request.setAttribute("user", user);
        request.setAttribute("rmbZhxx", rmbZhxx);
        request.setAttribute("xlbZhxx", xlbZhxx);
        request.setAttribute("userid",userid);
        forwardView(request, response, getClass());
    }
}
```





查看用户充值记录代码如下：

```
@Right(id = "C_USER_LIST_CK", name = "用户列表-查看")
public class ViewCzjl extends AbstractUserServlet {

    private static final long serialVersionUID = 1L;

    @Override
    protected void processGet(HttpServletRequest request,
        HttpServletResponse response, ServiceSession serviceSession)
        throws Throwable {
        processPost(request, response, serviceSession);
    }
    @Override
    protected void processPost(final HttpServletRequest request,
        final HttpServletResponse response,
        final ServiceSession serviceSession) throws Throwable {
        UserManager userManager = serviceSession.getService(UserManager.class);
        int userid = IntegerParser.parse(request.getParameter("userid"));
        User user=userManager.get(userid);
        AccountInfoEntity cnyAccountInfo=userManager.CNYAccountInfo(userid);
        PagingResult<CzglRecord> result =
userManager.getUserRechargeRecordList(new CzglRecordQuery() {
            @Override
            public ChargeStatus getStatus() {
                return EnumParser.parse(ChargeStatus.class, request.getParameter
("status"));
            }
            @Override
            public Timestamp getStartRechargeTime() {
                return TimestampParser.parse(request.getParameter("startRechargeTime"));
            }
            ..... // 省略部分字段
            @Override
            public int getUserid() {
                return IntegerParser.parse(request.getParameter("userid"));
            }
        }, new Paging() {
            @Override
            public int getSize() {
                return 10;
            }
        });
    }
}
```



```

    }
    @Override
    public int getCurrentPage() {
        return IntegerParser.parse(request.getParameter(PAGING_CURRENT));
    }
    },userid);
    request.setAttribute("user", user);
    request.setAttribute("cnyAccount", cnyAccount);
    request.setAttribute("userid",userid);
    request.setAttribute("result",result);
    forwardView(request, response, getClass());
}
}

```

查看用户提现记录的代码与查看充值记录相似，只是改变了查询类型。

### (3) 编辑用户信息

当需要修改某个用户的账户信息时，在搜索到该用户后，通过点击用户信息后面的“编辑”即可进入修改用户页面，如图 4-35 所示。该页面的主要功能是设置用户的账户权限。比如将“转币限额”设置为“0”，则该用户就没有转币限额了，可以无限制地转出资产；将“特殊账号”设置为“是”，则表明该账号为内部测试或运维人员账号；“免手续费”用于设置该用户在进行交易或转出资产时是否免除手续费。

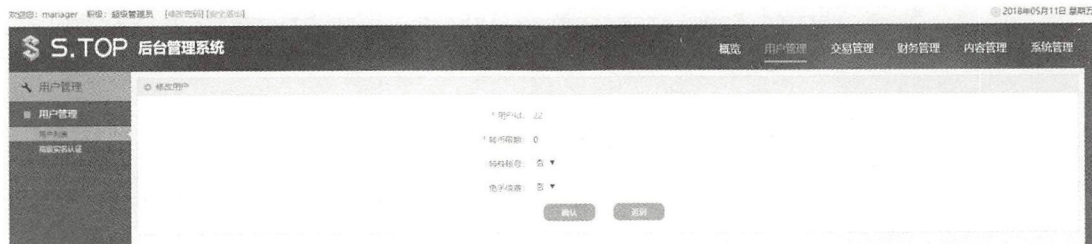


图 4-35

编辑用户信息代码如下：

```

@Right(id = "C_USER_UPDATE", name = "用户列表-编辑")
public class UpdateUser extends AbstractUserServlet {

    private static final long serialVersionUID = 1L;

    @Override

```





```
protected void processGet(HttpServletRequest request, HttpServletResponse
response, ServiceSession serviceSession) throws Throwable {
    UserManager userManager = serviceSession.getService(UserManage.class);
    int userid = IntegerParser.parse(request.getParameter("userid"));
    User user = userManager.get(userid);
    request.setAttribute("user", user);
    forwardView(request, response, getClass());
}

@Override
protected void processPost(final HttpServletRequest request, final
HttpServletResponse response, final ServiceSession serviceSession) throws
Throwable {
    UserManager userManager = serviceSession.getService(UserManage.class);
    int userid = IntegerParser.parse(request.getParameter("userid"));
    userManager.updateUser(new UserQuery() {
        @Override
        public String getIDNum() {
            return request.getParameter("IDNum");
        }

        @Override
        public IsPass getIsSpecialAccount() {
            return EnumParser.parse(IsPass.class, request.getParameter
("isSpecial"));
        }

        @Override
        public int getTransOutQuota() {
            return IntegerParser.parse(request.getParameter("TransOutQuota"));
        }

        @Override
        public IsPass getNofee() {
            return EnumParser.parse(IsPass.class, request.getParameter("nofee"));
        }
    }, userid);
    sendRedirect(request, response, getController().getURI(request,
UserList.class));
}
}
```





更新 SQL 语句如下:

```
execute(getConnection(P2PConst.DB_USER), "UPDATE UserAccount SET is_
specialAccount = ?,TransOutLimit = ?,is_nofee = ? WHERE accountId = ?", query.
getis_specialAccount(),query.getTransOutLimit(),query.getNofee(),userid);
execute(getConnection(P2PConst.DB_USER), "UPDATE UserInfo SET name=?,
CertificateType=?,IDnum=? WHERE accountId=?", query.getName(),query.
getCertificateType(), query.getZjh(), userid);
```

## 2. 用户黑白名单

当某个用户的账号出现异常需要对该账号进行锁定时,在搜索到该用户信息后,点击用户信息后面的“锁定”即可锁定该账号,如图 4-36 所示。账号被锁定后,就无法再进行登录和交易了。该锁定动作可以由后台人员手动完成,也可以由该系统的对账系统来进行自动锁定(请参见 4.2.5 节)。如果要解锁账号,只需点击“解锁”即可完成。

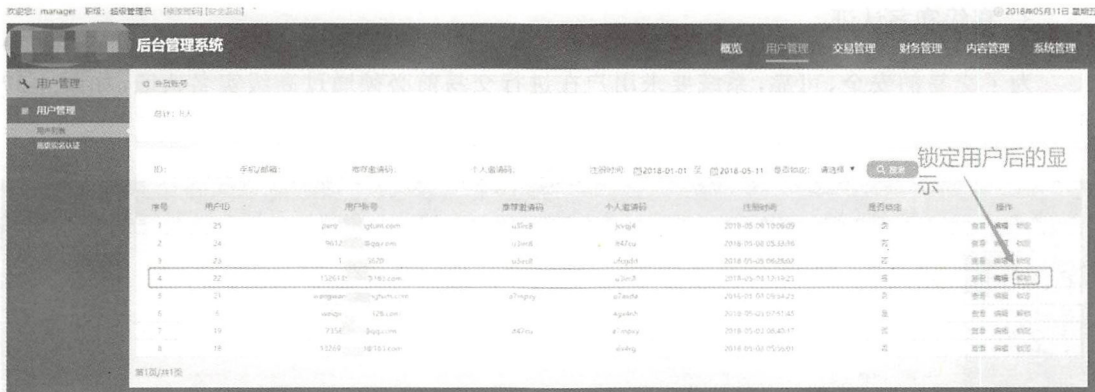


图 4-36

锁定账号代码如下:

```
@Right(id = "C_USER_LOCK", name = "用户列表-锁定账号")
public class UserLock extends AbstractUserServlet {
    private static final long serialVersionUID = 1L;
    @Override
    protected void processGet(HttpServletRequest request, HttpServletResponse
response, ServiceSession serviceSession) throws Throwable {
        processPost(request, response, serviceSession);
    }
}
```



```

@Override
protected void processPost(final HttpServletRequest request, final
HttpServletRequest response,final ServiceSession serviceSession) throws
Throwable {
    UserManager userManager = serviceSession.getService(UserManage.class);
    int userid = IntegerParser.parse(request.getParameter("userid"));
    IsPass isLock=EnumParser.parse(IsPass.class, request.getParameter
("isLock"));
    userManager.lockUser(isLock, userid);
    sendRedirect(request, response, getController().getURI(request,
UserList.class));
}
}

```

在“解锁”账号时，只需将锁定状态由“S”改为“F”即可。

### 3. 高级实名认证

为了交易的安全、可靠，系统要求用户在进行交易前必须通过高级实名认证，如图 4-37 所示。

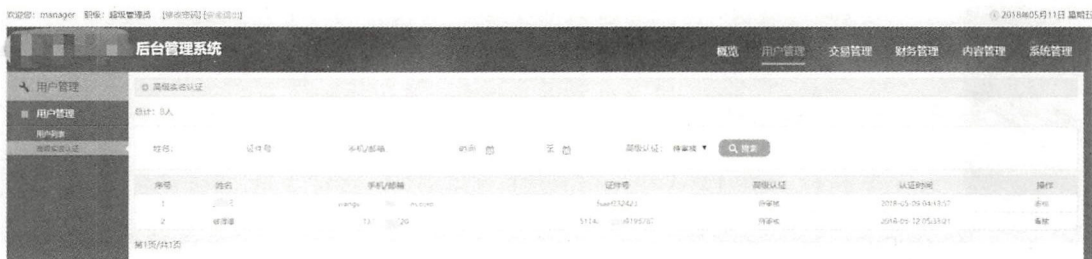


图 4-37

当用户在前台页面中进行高级实名认证，并提交身份证信息后，后台人员可以在此页面中对用户所提交的身份证图片信息进行审核，找到相应的用户信息，点击“审核”进入用户身份信息的审核页面，如图 4-38 所示。

后台人员审核完成后，点击“通过”或“不通过”，即可更新该用户的高级认证状态。当点击“不通过”时，可以选择没有通过认证的原因，比如正面照片不清晰、反面照片不清晰、年龄不符合规定或其他等。下面有具体原因描述输入框，可以输入没有通过认证的具体原因。点击“确定”后，该审核结果会以邮件或短信的形式发送给用户。



图 4-38

高级实名认证审核代码如下：

```
@Right(id = "C_USER_LIST_SH", name = "高级实名认证-审核")
public class UserGjrzSh extends AbstractUserServlet {
    private static final long serialVersionUID = 1L;
    @Override
    protected void processGet(HttpServletRequest request, HttpServletResponse
response, ServiceSession serviceSession) throws Throwable {
        int userid = IntegerParser.parse(request.getParameter("userid"));
        UserManager userManager = serviceSession.getService(UserManager.class);
        User user = userManager.get(userid);
        request.setAttribute("user", user);
        forwardView(request, response, getClass());
    }
    @Override
    protected void processPost(final HttpServletRequest request, final
HttpServletResponse response, final ServiceSession serviceSession) throws
Throwable {
        UserManager userManager = serviceSession.getService(UserManager.class);
        int userid = IntegerParser.parse(request.getParameter("userid"));
        IsPass type = EnumParser.parse(IsPass.class, request.getParameter
("type"));
        String remark = request.getParameter("remark");
        // 备注=不通过类型+具体原因
        if("其他".equals(remark)){
            remark = request.getParameter("content");
        }
        userManager.gjrzSh(userid, type, remark);
    }
}
```





```
sendRedirect(request, response, getController().getURI(request,
    UserAdvancedCertification.class));
}
```

## 4.2.2 交易管理

交易管理主要包括虚拟币管理、分区管理、市场管理、委托管理、成交记录、资产积分转入统计等功能，提供了代币、创建交易分区及市场、查看委托信息、撮合成功后查看成交记录，以及用户向平台转入的资产积分统计记录等。

### 1. 虚拟币管理

虚拟币管理的主要功能就是查询、添加、修改和删除虚拟币，如图 4-39 所示。

后台管理系统

交易管理

ID	英文名称	中文名	图标	币种	正常转入	正常转出	最小限额	最大限额	最近转出手续费	转出最高手续费率	自动转入	自动转入值	时间	操作
1	Bitcoin	比特币		Bitcoin	10000000	10000000	0.000001	10000000	0.000001	0.000001	是	0.000001	2018-05-15 11:11:11	编辑 删除
2	Ether	以太坊		Ether	10000000	10000000	0.000001	10000000	0.000001	0.000001	是	0.000001	2018-05-15 11:11:11	编辑 删除
3	Bitcoin Cash	比特币现金		Bitcoin Cash	10000000	10000000	0.000001	10000000	0.000001	0.000001	是	0.000001	2018-05-15 11:11:11	编辑 删除
4	Bitcoin SV	比特币现金		Bitcoin SV	10000000	10000000	0.000001	10000000	0.000001	0.000001	是	0.000001	2018-05-15 11:11:11	编辑 删除
5	Bitcoin Gold	比特币现金		Bitcoin Gold	10000000	10000000	0.000001	10000000	0.000001	0.000001	是	0.000001	2018-05-15 11:11:11	编辑 删除
6	Bitcoin Cash ABC	比特币现金		Bitcoin Cash ABC	10000000	10000000	0.000001	10000000	0.000001	0.000001	是	0.000001	2018-05-15 11:11:11	编辑 删除

图 4-39

#### (1) 添加虚拟币

只有添加了虚拟币，前台页面才可以为该币种生成钱包地址，用户才可以转入或买入该种类型的虚拟币，在币种下拉列表框中才会出现该币种。因此，添加虚拟币是整个交易系统开始的步骤。

添加虚拟币的页面如图 4-40 所示。

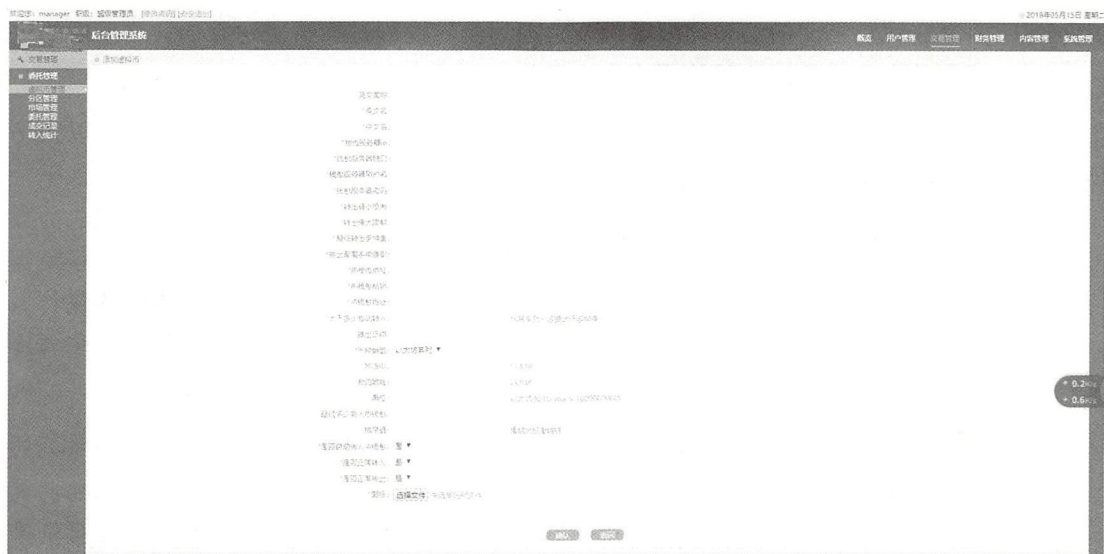


图 4-40

添加虚拟币的必填信息还是比较多的。在添加虚拟币前，需要准备好币种的正式名称信息、币种图标、钱包服务器信息（IP 地址、端口、用户名、密码）、转入/转出及交易的限制（如：转出最小量或最大量、转出手续费）等。

- ◎ 英文简称、英文名、中文名：在上币前最好到官方网站查询。
- ◎ 钱包服务器的 IP 地址、端口、用户名、密码：填写这些信息主要是为了转入资产积分时为用户创建临时钱包地址。当用户将资产转入临时钱包后，该资产会自动转入平台上该币种的冷钱包中。这样做是出于对用户资产安全的考虑。
- ◎ 转出最小限额、转出最大限额和转出手续费：这些信息是平台根据自己的需求来设定的。
- ◎ 热钱包地址、热钱包私钥、冷钱包地址：这两个钱包地址是事先申请好并已经激活了的，可以通过钱包地址生成器来生成相应币种的钱包地址。一般每个币种需要申请三个钱包地址，即热钱包、冷钱包和提币钱包。当用户向临时钱包转入资产积分后，临时钱包中的资产会自动转入热钱包，再从热钱包转入平台冷钱包。因此，这里需要填写热钱包的地址和私钥，以及冷钱包的地址。
- ◎ 大于多少自动转入：当热钱包中的资产大于某个值后自动转入冷钱包。
- ◎ 超过多少转入热钱包：当用户向临时钱包转入的资产大于某个值时自动转入热钱包。



- ◎ 资产在钱包之间的流转需要手续费，因为流转过程是在链上进行的。这里面的手续费是由公司承担还是让用户个人承担，则视公司情况而定。

不同币种收取的手续费是不同的，若一个币种为这个系列币种的法币，如以太坊系列的 ETH、井通系列的 SWTC 等，则要求用户在将自己的资产首次转入平台为其生成的临时钱包前，应先激活该临时钱包，并且需要转入大于激活钱包的数额的法币资产。因为用户资产转入临时钱包后会自动转入热钱包，这一过程是需要消耗手续费的，该手续费即临时钱包里的法币。因此，当资产从临时钱包转入热钱包时，临时钱包里必须有一定数量的该法币。同理，当资产由热钱包转入冷钱包时，热钱包中也必须有一定数量的该法币作为手续费。

若该币种为所属币种系列的代币，如以太坊系列的 DABT、MOAC，井通系列的 WGC、DDB 等，则在用户将资产积分首次转入平台为其生成的临时钱包前，需要先充值其所属币种系列的法币将钱包激活后，才能正常转入该代币。而且资产从临时钱包到热钱包再到冷钱包所收取的一系列手续费也是该币种系列的法币。因此，当资产在钱包之间流转时，各个钱包里面应该有一应数量的该币种系列的法币作为手续费。

- ◎ 币种类型：选择待添加的币种所属的币种系列。若为以太坊系列，则此处还应该添加以太坊系列币的方法 ID、合约地址及单位。
- ◎ 选择币种是否正常转入、转出及币种的图标。

以上信息都填写完成后点击“确认”按钮，即可完成添加虚拟币。

添加虚拟币代码如下：

```
@MultipartConfig
@Right(id = "C_TRADING_ADDXLB", name = "虚拟币管理-添加")
public class AddToken extends AbstractTradingServlet {
    private static final long serialVersionUID = 1L;
    @Override
    protected void processPost(final HttpServletRequest request, final
    HttpServletResponse response, final ServiceSession serviceSession) throws
    Throwable {
        TradingManage tsradingManage = serviceSession.getService
        (TradingManage.class);
        TokenQuery query = new TokenQuery() {
            @Override
```





```

        public BigDecimal getForwardLimit() {
            return BigDecimalParser.parse(request.getParameter("forwardLimit"));
        }
        @Override
        public String getChinese() {
            return request.getParameter("Chinese");
        }
        @Override
        public BigDecimal getMaxServiceCharge() {
            return BigDecimalParser.parse(request.getParameter
("MaxServiceCharge"));
        }
        @Override
        public BigDecimal getMinServiceCharge() {
            return BigDecimalParser.parse(request.getParameter
("MinServiceCharge"));
        }
        @Override
        public String getEnglish() {
            return request.getParameter("english");
        }
        @Override
        public String getWalletIp() {
            return request.getParameter("wallet_ip");
        }
        @Override
        public String getWalletPort() {
            return request.getParameter("wallet_port");
        }
        ..... // 省略其余字段, 请自行设计编写
    }
}

```

## (2) 编辑虚拟币信息

当手续费或钱包地址等发生变化需要调整该虚拟币的信息时, 点击“编辑”即可进入虚拟币编辑页面。首先获取虚拟币信息, 代码如下:

```

@Override
protected void processGet(HttpServletRequest request, HttpServletResponse
response, ServiceSession serviceSession) throws Throwable {
    int id=IntegerParser.parse(request.getParameter("id"));
}

```



```

        TradingManage tradingManage = serviceSession.getService
        (TradingManage.class);
        TokenEntity token=tradingManage.getToken(id);
        request.setAttribute("token", token);
        forwardView(request, response, getClass());
    }

```

编辑完成后点击“确认”按钮，即可更新该虚拟币信息。在前台页面中对该币种的设定也随之改变。编辑虚拟币信息代码如下：

```

@MultipartConfig
@Right(id = "C_TRADING_UPDATEEXLB", name = "虚拟币管理-编辑")
public class updateToken extends AbstractTradingServlet {
    private static final long serialVersionUID = 1L;
    @Override
    protected void doPost(final HttpServletRequest request, final
    HttpServletResponse response, final ServiceSession serviceSession) throws
    Throwable {
        TradingManage tradingManage = serviceSession.getService
        (TradingManage.class);
        int id=IntegerParser.parse(request.getParameter("id"));
        TokenQuery query = new TokenQuery() {
            @Override
            public BigDecimal getForwardLimit() {
                return BigDecimalParser.parse(request.getParameter
                ("forwardLimit"));
            }
            @Override
            public BigDecimal getMaxServiceCharge() {
                return BigDecimalParser.parse(request.getParameter
                ("MaxServiceCharge"));
            }
            @Override
            public BigDecimal getMinServiceCharge() {
                return BigDecimalParser.parse(request.getParameter
                ("MinServiceCharge"));
            }
            @Override
            public String getWalletIp() {
                return request.getParameter("wallet_ip");
            }
        };
    }

```



```
    }  
    @Override  
    public String getWalletPort() {  
        return request.getParameter("wallet_port");  
    }  
    .....// 省略其余字段，请自行设计编写  
}  
}
```

### (3) 删除虚拟币

当该币种不再需要时，可以点击“删除”来删除该币种。删除后该币种在前台和后台的各个币种下拉列表框中将不再出现。

删除虚拟币代码如下：

```
@Right(id = "C_TRADING_DELETEXLB", name = "虚拟币管理-删除")  
public class DeleteXlb extends AbstractTradingServlet {  
  
    private static final long serialVersionUID = 1L;  
    @Override  
    protected void processPost(final HttpServletRequest request, final  
    HttpServletResponse response, final ServiceSession serviceSession) throws  
    Throwable {  
        TradingManage tradingManage = serviceSession.getService  
    (TradingManage.class);  
        int id=IntegerParser.parse(request.getParameter("id"));  
        tradingManage.deleteToken(id);  
        sendRedirect(request, response, getController().getURI(request,  
    TokenList.class));  
    }  
}
```

### (4) 链接

点击链接会跳转到“常用链接管理”页面，如图 4-41 所示。该页面的参数设置信息最终会显示到前台的“币币交易”中的该币种的“了解币种”页面。

点击“添加”即可添加一个常用链接，包括语言、名称、链接地址、排序和是否显示等，如图 4-42 所示。添加成功后，还可以编辑或删除该币种的常用链接。





图 4-41

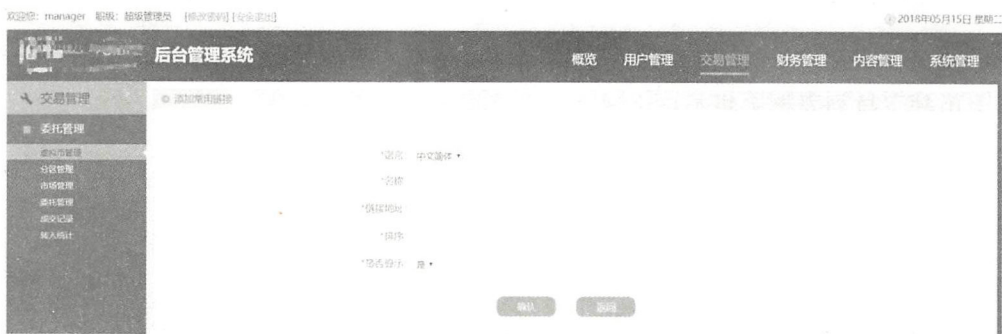


图 4-42

### (5) 参数

常用参数管理的作用与常用链接管理相似，都是为了让用户了解该币种，如图 4-43 所示。关于具体的添加、修改、删除常用参数操作这里不再赘述。



图 4-43

当常用链接和常用参数设置完成后，前台页面的显示效果如图 4-44 所示。

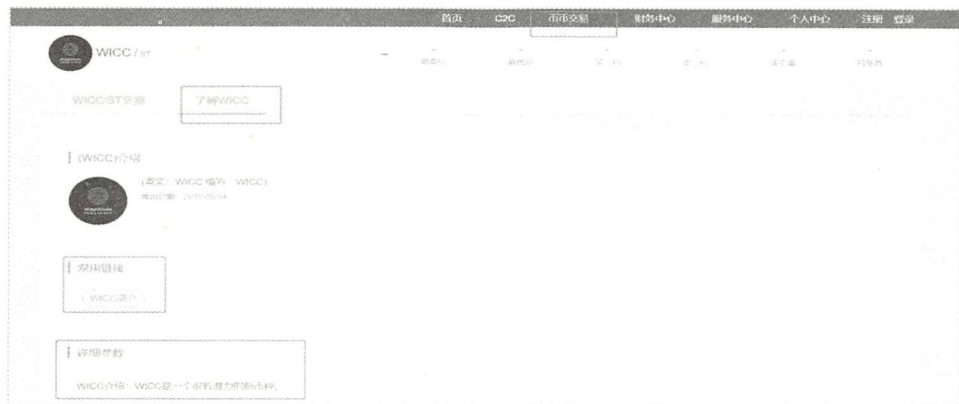


图 4-44

通过常用链接和详细参数，用户可以在很短的时间内了解该币种的主要信息及价值。

#### (6) 查询

当币种变多了以后，我们查找一个币种，如果一页一页逐个查找将会很费时，而虚拟币列表页面支持对虚拟币的条件查询，如通过币种名称或添加虚拟币的时间段来进行查询。

条件查询虚拟币列表代码如下：

```
@Right(id = "C_TRADING_XLBLIST", isMenu = true, name = "虚拟币管理")
public class TokenList extends AbstractTradingServlet {
    private static final long serialVersionUID = 1L;
    @Override
    protected void processGet(HttpServletRequest request, HttpServletResponse
response, ServiceSession serviceSession) throws Throwable {
        processPost(request, response, serviceSession);
    }
    @Override
    protected void processPost(final HttpServletRequest request, final
HttpServletResponse response, final ServiceSession serviceSession) throws
Throwable {
        TradingManage tradingManage = serviceSession.getService
(TradingManage.class);
        PagingResult<TokenEntity> tokens = tsradingManage.tokenSearch(new
TokenQuery() {

            @Override
            public String getChinese() {
```



```

        return request.getParameter("Chinese");
    }
    @Override
    public Timestamp getCreateTimeStart() {
        return TimestampParser.parse(request.getParameter("start"));
    }
    @Override
    public Timestamp getCreateTimeEnd() {
        return TimestampParser.parse(request.getParameter("end"));
    }
    }, new Paging() {
        @Override
        public int getSize() {
            return 10;
        }
        @Override
        public int getCurrentPage() {
            return IntegerParser.parse(request.getParameter(PAGING_CURRENT));
        }
    });
    request.setAttribute("tokens", tokens);
    forwardView(request, response, getClass());
}
}

```

## 2. 分区管理

添加完虚拟币后，就可以新建分区了。分区的作用是划分不同系列币种的交易场所，如以太坊系分区、井通系分区等，如图 4-45 所示。

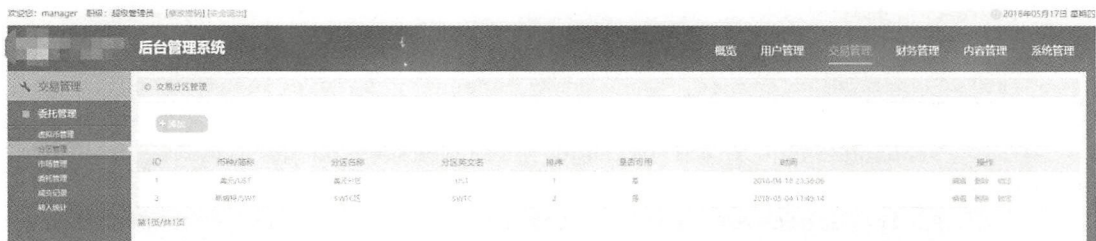


图 4-45





交易分区的数据库表结构如下:

```
CREATE TABLE `SubareaInfo` (  
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '自增 ID',  
  `coinId` int(10) DEFAULT NULL COMMENT '币种 ID;参考 CoinInfo.id',  
  `subareaNameCn` varchar(10) DEFAULT NULL COMMENT '分区名称',  
  `subareaNameEn` varchar(10) DEFAULT NULL COMMENT '分区英文名',  
  `is_usable` enum('S','F') DEFAULT 'S' COMMENT '是否可用',  
  `rank` int(11) DEFAULT '0' COMMENT '排序',  
  `createTime` timestamp NULL DEFAULT NULL COMMENT '操作时间',  
  `is_delete` enum('F','S') DEFAULT 'F' COMMENT '是否删除',  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8 COMMENT='交易分区信息 (SubareaInfo)';
```

### (1) 添加分区

如图 4-46 所示,添加分区比较简单,只需要四步即可完成。首先在“币种”下拉列表框中选择之前添加的虚拟币,然后填写分区名称和分区英文名称,接下来选择序号,序号越大则这个分区越靠前。最后选择是否可用,选择“是”后,该分区会被显示在前台的“币币交易”页面中。

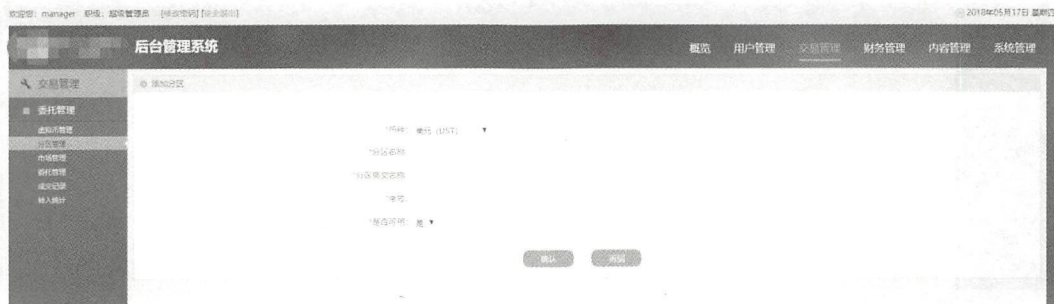


图 4-46

添加分区代码如下:

```
@Override  
public void addTransPartition(TransPartitionQuery query) throws Throwable {  
  if (query != null) {  
    if (query.getRank() <= 0) {  
      throw new ParameterException("排序值不能小于 0。");  
    }  
    if (StringHelper.isEmpty(query.getPartitionName())) {
```



```

        throw new ParameterException("分区名称不能为空。");
    }
    if (StringHelper.isEmpty(query.getPartitionNameEn())) {
        throw new ParameterException("分区英文名称不能为空。");
    }
}

execute(getConnection(P2PConst.DB_USER), "INSERT INTO PartitionInfo SET
coinId=?,partitionName=?,partitionNameEn=?,is_usable=?,rank=?, createTime=
CURRENT_TIMESTAMP()", query.getCoinId(), query.getPartitionName(), query.
getPartitionNameEn(), query.is_usable(), query.getRank());
}

```

添加分区完成后，回到分区列表页面，此时我们可以看到在每个分区信息后面有三个选项，即“编辑”“删除”和“锁定”。

## (2) 编辑分区

当分区信息需要变更时，我们可以选择“编辑”选项，进入该分区的编辑页面，如图4-47所示。

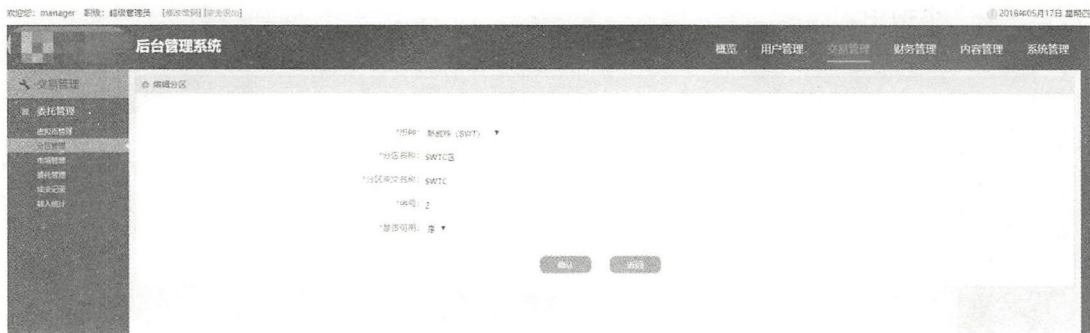


图 4-47

更改分区信息，然后点击“确认”按钮即可。前台分区信息的设置会随之改变。

编辑分区代码如下：

```

@Right(id = "C_TRADING_UPDATEJYFQ", name = "分区管理-编辑")
public class UpdateTransPartition extends AbstractTradingServlet {
    private static final long serialVersionUID = 1L;
    @Override
    protected void processGet(HttpServletRequest request, HttpServletResponse

```



```

response, ServiceSession serviceSession) throws Throwable {
    TradingManage tradingManage = serviceSession.getService
(TradingManage.class);
    int id = IntegerParser.parse(request.getParameter("id"));
    TransPartitionEntity transPart=tradingManage.getTransPartition(id);
    TokenEntity [] token=tradingManage.getTokenInfo();
    request.setAttribute("transPart", transPart);
    request.setAttribute("token", token);
    forwardView(request, response, getClass());
}

@Override
protected void processPost(final HttpServletRequest request, final
HttpServletResponse response, final ServiceSession serviceSession) throws
Throwable {
    int id = IntegerParser.parse(request.getParameter("fqid"));
    TradingManage tradingManage = serviceSession.getService
(TradingManage.class);
    tradingManage.updateTransPartition(new TransPartitionQuery() {
        @Override
        public IsPass is_usable() {
            return EnumParser.parse(IsPass.class, request.getParameter
("is_usable"));
        }
        @Override
        public int getRank() {
            return IntegerParser.parse(request.getParameter("rank"));
        }
        ..... // 省略数个方法
    }, id);
    sendRedirect(request, response, getController().getURI(request,
TransPartList.class));
}
}

```

### (3) 删除分区

当分区不再使用后,可以选择“删除”分区,删除后该分区在前台页面中会随之删除。删除分区是一件比较慎重的事情,尤其是如果之前已经上线使用,而且在该分区内的市场下还有很多委托挂单,若突然删除分区,会造成很多不必要的麻烦,因此最好在删除之前,保证该分区下的所有交易已完成、所有的委托挂单已完成或已批量撤销,然后再进行分区





的删除操作。

删除分区代码如下：

```
@Override
public void deleteTranSubarea(int subareaid) throws Throwable {
    if (subareaid <= 0) {
        throw new ParameterException("ID 异常。");
    }
    execute(getConnection(P2PConst.DB_USER), "UPDATE SubareaInfo SET
is_delete=? WHERE subareaId=?", IsPass.S.name(), subareaid);
}
```

通过实现的 SQL 可以看到，在删除分区时，并没有真正地删除分区，而是将该分区的“是否删除”状态由“F”改成了“S”。当在分区列表中进行分区信息查询时，如果查询到该分区信息的“是否删除”状态为“S”，则不再查询和显示该分区信息，这样被删除的分区就不会显示在前台的“币币交易”页面中了。

交易分区列表查询代码如下：

```
@Override
public PagingResult<TransPartitionEntity> transPartitionSearch(Paging
paging) throws Throwable {
    return selectPaging(getConnection(P2PConst.DB_USER), new ArrayParser
<TransPartitionEntity>()) {
        @Override
        public TransPartitionEntity[] parse(ResultSet re) throws
SQLException {
            ArrayList<TransPartitionEntity> list = new ArrayList
<TransPartitionEntity>();
            while (re.next()) {
                TransPartitionEntity x = new TransPartitionEntity();
                x.id = re.getInt(1);
                x.coinId = re.getInt(2);
                x.partitionName = re.getString(3);
                x.partitionNameEn = re.getString(4);
                x.isPass = EnumParser.parse(IsPass.class, re.getString(5));
                x.rank = re.getInt(6);
                x.time = re.getTimestamp(7);
                x.simpleNameEn = re.getString(8);
                x. = re.getString(9);
                if (list == null) {
```



```
        list = new ArrayList<>();
    }
    list.add(x);
}
return list == null ? null : list.toArray(new
TransPartitionEntity[list.size()]);
}
}, paging,
"SELECT TransPartition.id,TransPartition.coinId coinId,TransPartition.
partitionName, TransPartition.partitionNameEn partitionNameEn ,TransPartition.
is_usable,TransPartition.rank, TransPartition.createTime, CoinInfo.coinNameEn
coinNameEn, CoinInfo.coinNameCn coinNameCn FROM TransPartition LEFT JOIN
CoinInfo ON TransPartition.coinId = CoinInfo.id WHERE TransPartition.is_delete
= ? ORDER BY CoinInfo.is_normalRollIn DESC CoinInfo.createTime DESC
",IsPass.F.name());
}
```

我们可以看到，在查询 SQL 中有一个判断条件 WHERE TransPartition.is\_delete = ? IsPass.F.name()，查询的是所有分区中“是否删除”状态为“F”的分区。因此，想要重新使用这个分区，只需要将数据库中该分区的“是否删除”状态由“S”改为“F”，即可重新使用该分区，也不用再次新建。

我们不彻底删除分区的目的不是为了重新使用它，是为了保留这个分区的信息。比如可以通过分区 ID 来查询这个分区下的所有市场，以及市场下的所有交易记录。如果彻底删除分区，以后在查询时就可能会发现某条交易记录的市场是空白的，这就造成了不必要的麻烦。同理，我们在删除某个功能时应该考虑，是彻底删除还是只更改它的状态。

#### （4）锁定分区

如果要暂停某分区下的所有交易，则可以锁定该分区。锁定之后的分区不会显示在前台的“币种交易”页面中，而且所有挂出去的委托订单也会暂停撮合。

### 3. 市场管理

当分区创建完成后，就可以创建市场了。市场的作用是具体规定在该市场内哪两个币种可以进行互相交易，哪个币种是买方，哪个币种是卖方。因此创建市场较为烦琐，也比较考验一个人的细心。首先看一下创建好的市场列表，如图 4-48 所示。







太小，则可能会导致资产显示不全。紧接着填写买入/卖出手续费率，这是由公司运营人员来决定的。手续费的收取原则是：得到什么，手续费就收取什么。比如用户买入的是 SWTC，当买到 10 个 SWTC 后，平台会收取手续费，手续费即用户买入的 SWTC 个数乘以手续费率。同理，若用户是卖家，卖出 SWTC 得到了 5 个 ETH，则所收取的手续费即用户得到的 ETH 乘以手续费率。

输入买入/卖出最小交易价和最大交易价。交易价是指当买家或卖家需要买入或卖出某个币种时，他们会先输入想以单个多少的价格来买入或卖出，然后输入买入或卖出的数量，输入完成确定后平台会计算这次交易的总额。这个总额是不能小于在此处设定的最小交易价或大于在此处设定的最大交易价的。

接着选择“是否启用最大成交量限制”。“最大成交量限制”是指限制每个用户每日的最大卖出成交量和最大买入成交量，以及市场每日的最大卖出成交量和最大买入成交量。这样限制是为了防止一些别有用心的用户故意抛售或买断某些币种，也为平台的资金安全设置了一层保护，保证不会在短时间内出现大量用户抛售货币跑路的情况，给平台一个预警信息和缓冲的时间。如果选择不开启最大成交量限制，那么它下面的四个输入框可以为空。

最后输入“排序值”，排序值越大，市场越靠前。选择“是否开启交易”，若选择开启交易，则它下面的“是否显示”会自动设置为“是”，反之则为“否”。

所有的数据设置完成后点击“确认”按钮，就可以新建一个市场。

新建市场实现类方法如下：

```
@Override
public void addTradingMarket (TradingMarket Query query) throws Throwable {
    if (query != null) {
        if (query.getSubareaId() <= 0) {
            throw new ParameterException("分区 ID 不能小于 0。");
        }
        if (query.getSell_coinId() <= 0) {
            throw new ParameterException("卖方币种 ID 不能小于 0。");
        }
        if (query.getDecimals() < 0) {
            throw new ParameterException("小数位数不能小于 0。");
        }
        if (query.getBuy_rate().compareTo(new BigDecimal(0)) < 0) {
            throw new ParameterException("买入手续费率不能小于 0。");
        }
        if (query.getSell_rate().compareTo(new BigDecimal(0)) < 0) {
```



```
        throw new ParameterException("卖出手续费率不能小于 0。");
    }
    if (query.getBuy_minPrice().compareTo(new BigDecimal(0)) < 0) {
        throw new ParameterException("买入最小交易价不能小于 0。");
    }
    ..... // 省略部分代码
}

execute(getConnection(P2PConst.DB_USER), "INSERT INTO MarketInfo SET
subareaId=?,buyCoin=?,sellCoin=?,Decimals=?,buyRate=?,sellRate=?,buyMinPrice
=?,buyMaxPrice=?,sellMinPrice=?,sellMaxPrice=?,is_openTrans=?,is_show=?,crea
teTime=CURRENT_TIMESTAMP(),userbuyMin=?,userSellMax=?,marketBuyMin=?,market
SellMax=?,is_openPosition Limit=?,rank=?", query.getsubareaId(),
query.getBuyCoin(), query.getSellCoin(), query.getDecimals(),
query.getBuyRate(), query.getSellRate(), query.getbuyMinPrice(),
query.getbuyMaxPrice(), query.getSellMinPrice(), query.getSellMaxPrice(),
query.getIs_openTrans(), query.getIs_openTrans(), query.getUserbuyMin(),
query.getUserSellMax(), query.getMarketBuyMin(), query.getMarketSellMax(),
query.getIs_openPosition Limit(), query.getRank());
}
```

## (2) 编辑市场

当需要对市场的一些设置参数进行变更时，可以选择“编辑”市场，进入编辑交易市场页面，修改需要变更的参数，如图 4-50 所示。一般变更手续费率、最大/最小交易价和是否启用最大成交量限制信息的概率比较大。编辑完成后点击“确认”按钮即可。

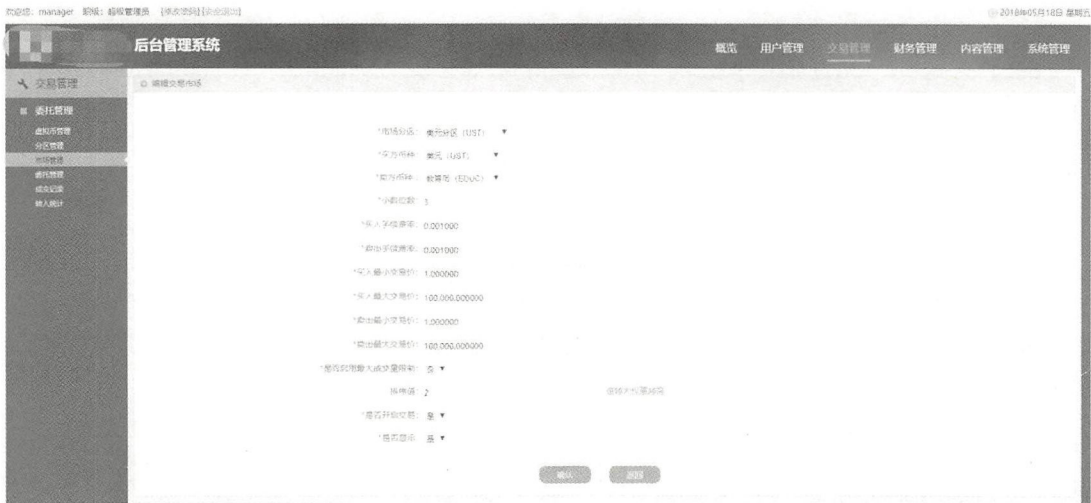


图 4-50



### (3) 删除市场

当需要删除市场时, 点击“删除”即可。

删除市场代码如下:

```
@Right(id = "C_TRADING_DELETEJYSC", name = "市场管理-删除")
public class DeleteTransMarket extends AbstractTradingServlet {
    private static final long serialVersionUID = 1L;
    @Override
    protected void processPost(final HttpServletRequest request, final
    HttpServletResponse response, final ServiceSession serviceSession) throws
    Throwable {
        TradingManage tradingManage = serviceSession.getService
        (TradingManage.class);
        int id = IntegerParser.parse(request.getParameter("id"));
        tradingManage.deleteTransMarket(id);
        sendRedirect(request, response, getController().getURI(request,
        TranMarketList.class));
    }
}
```

删除市场实现类方法如下:

```
@Override
public void deleteTransMarket(int id) throws Throwable {
    if (id <= 0) {
        throw new ParameterException("ID 异常。");
    }
    execute(getConnection(P2PConst.DB_USER), "UPDATE MarketInfo SET
    is_delete=? WHERE id=?", IsPass.S, id);
}
```

这里的“删除”市场并不是真正的删除, 就同删除分区一样, 只是将其删除状态由“F”改成了“S”, 同时保留了该市场 ID 及信息。在前台的“币币交易”的分区下显示市场时, 可以查询到该市场的状态为已删除, 但此时若该市场的“是否显示”状态为“是”, 那么它仍会显示在“币币交易”的相应分区下, 且其交易记录是可见的。在后台的市场列表中是查询不到该市场的, 但是在后台数据库中通过市场 ID 可以查询到该市场曾经的交易记录。在删除市场前, 应该将该市场的状态更改为不开启交易, 再批量撤销该市场下仍然存在的用户委托记录, 将冻结的用户资产返回到用户账户中。最后, 在确定没有问题之后,





再执行市场的删除操作。

#### (4) 隐藏市场

在删除市场后，被删除的市场不再显示在后台的市场列表中，但是在前台的“币币交易”的相应分区下仍然会显示该市场，而且该市场的所有历史成交记录也仍然存在。若想在前台也不显示被删除的市场，则可以选择隐藏市场。点击“隐藏”后，在前台的“币币交易”的分区下显示市场时查询到该市场已隐藏，则不会再显示该市场，而不论该市场有没有被删除，或者是否开启了交易。因此在隐藏或删除市场前，应首先关闭该市场下的交易，然后考虑是否显示，最后再执行隐藏或删除市场的操作。

数据库市场表结构如下：

```
CREATE TABLE `marketInfo` (  
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '自增 ID',  
  `subareaId` int(10) DEFAULT NULL COMMENT '分区 ID;参考 SubareaInfo.id',  
  `buy_coin` int(10) DEFAULT NULL COMMENT '买方币种;参考 CoinInfo.id(购买币种)',  
  `sell_coin` int(10) DEFAULT NULL COMMENT '卖方币种;参考 CoinInfo.id(商品币种)',  
  `decimals` int(11) DEFAULT '0' COMMENT '小数位数',  
  `buy_rate` decimal(20,8) DEFAULT '0.00000000' COMMENT '买入手续费率',  
  `sell_rate` decimal(20,8) DEFAULT '0.00000000' COMMENT '卖出手续费率',  
  `buy_minPrice` decimal(20,8) DEFAULT '0.00000000' COMMENT '买入最小交易价',  
  `buy_maxPrice` decimal(20,8) DEFAULT '0.00000000' COMMENT '买入最大交易价',  
  `sell_minPrice` decimal(20,8) DEFAULT '0.00000000' COMMENT '卖出最小交易价',  
  `sell_maxPrice` decimal(20,8) DEFAULT NULL COMMENT '卖出最大交易价',  
  `is_openTrans` enum('S','F') DEFAULT 'S' COMMENT '是否开启交易',  
  `is_show` enum('S','F') DEFAULT 'S' COMMENT '是否显示',  
  `createTime` timestamp NULL DEFAULT NULL COMMENT '时间',  
  `rank` int(10) DEFAULT NULL COMMENT '排序值',  
  `buy_minQuantity` decimal(20,2) DEFAULT NULL COMMENT '买入最小量',  
  `buy_maxQuantity` decimal(20,2) DEFAULT NULL COMMENT '买入最大量',  
  `sell_minQuantity` decimal(20,2) DEFAULT NULL COMMENT '卖出最小量',  
  `sell_maxQuantity` decimal(20,2) DEFAULT NULL COMMENT '卖出最大量',  
  `is_delete` enum('F','S') DEFAULT 'F' COMMENT '是否删除',  
  `userBuyMax` decimal(20,2) DEFAULT NULL COMMENT '用户日买入量-最大限制',  
  `userSellMin` decimal(20,2) DEFAULT NULL COMMENT '用户日卖出量-最大限制',  
  `marketBuyMax` decimal(20,2) DEFAULT NULL COMMENT '市场日买入量-最大限制',  
  `marketSellMin` decimal(20,2) DEFAULT NULL COMMENT '市场日卖出量-最大限制',  
  `is_openTransLimit` enum('S','F') DEFAULT 'F' COMMENT '是否启用最大成交量限制',  
  PRIMARY KEY (`id`)
```



```
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 COMMENT='交易市场
(marketInfo) ';
```

#### 4. 委托管理

在“委托管理”页面中可以看到平台所有分区及市场下的用户委托订单，包括委托 ID、用户名、市场、单价、数量、已成交、总额、类型、状态、时间等，如图 4-51 所示。



图 4-51

由于委托信息很多，查看起来十分麻烦，所以可以按条件查询。这里提供的条件最多，包括通过用户名、时间、交易市场、委托类型、状态来进行条件查询。

条件查询委托记录代码如下：

```
@Right(id = "C_TRADING_LIST", isMenu = true, name = "委托管理")
public class EntrustList extends AbstractTradingServlet {
    private static final long serialVersionUID = 1L;
    @Override
    protected void processGet(HttpServletRequest request, HttpServletResponse
response, ServiceSession serviceSession) throws Throwable {
        processPost(request, response, serviceSession);
    }
    @Override
    protected void processPost(final HttpServletRequest request, final
HttpServletResponse response, final ServiceSession serviceSession) throws
Throwable {
        TradingManage tradingManage = serviceSession.getService
(TradingManage.class);
        TransPartitionEntity[]
transPartition=tradingManage.transPartitionSearch();
        PagingResult<TransMarketEntity> tradings = tradingManage.tradingSearch
```





```
(new TradingQuery() {
    @Override
    public EntrustType getEntrustStatus() {
        return EnumParser.parse(EntrustType.class, request.getParameter
("entrustStatus"));
    }
    @Override
    public String getUsername() {
        return request.getParameter("userName");
    }
    @Override
    public TransStatus getTransStatus() {
        return EnumParser.parse(TransStatus.class, request.getParameter
("TransType"));
    }
    @Override
    public int getId() {
        return IntegerParser.parse(request.getParameter("id"));
    }
    @Override
    public Timestamp getCreateTimeStart() {
        return TimestampParser.parse(request.getParameter("start"));
    }
    @Override
    public Timestamp getCreateTimeEnd() {
        return TimestampParser.parse(request.getParameter("end"));
    }
}, new Paging() {
    @Override
    public int getSize() {
        return 10;
    }
    @Override
    public int getCurrentPage() {
        return IntegerParser.parse(request.getParameter(PAGING_CURRENT));
    }
});
request.setAttribute("tradings", tradings);
request.setAttribute("TransPartitions", TransPartitions);
forwardView(request, response, getClass());
}
```





选择查询条件后, 点击“搜索”按钮, 可以查看到相应条件下的委托记录。

### (1) 将委托记录导出为 Excel 文件

```
@Right(id = "C_TRADING_EXPORWT", name = "交易管理-委托管理-导出")
public class ExportEntrust extends AbstractFinancialServlet {
    private static final long serialVersionUID = 1L;
    @Override
    protected void processGet(HttpServletRequest request, HttpServletResponse
response, ServiceSession serviceSession) throws Throwable {
        processPost(request, response, serviceSession);
    }
    @Override
    protected void processPost(final HttpServletRequest request, final
HttpServletResponse response, final ServiceSession serviceSession) throws
Throwable {
        response.setHeader("Content-disposition", "attachment;filename=" +
new Timestamp(System.currentTimeMillis()).getTime() + ".csv");
        response.setContentType("application/csv");
        TradingManage tradingManage = serviceSession.getService
(TradingManage.class);
        PagingResult<TransMarketEntity> tradings = tradingManage.tradingSearch
(new TradingQuery() {
            @Override
            public EntrustType getEntrustStatus() {
                return EntrustType.InTrading;
            }
            @Override
            public String getUsername() {
                return request.getParameter("userName");
            }
            @Override
            public JystatusStatus getTransStatus() {
                return EnumParser.parse(TransStatus.class, request.getParameter
("TransStatus"));
            }
            ..... // 省略部分字段
        }, new Paging() {
            @Override
            public int getSize() {
                return Integer.MAX_VALUE;
            }
            @Override
```



```

        public int getCurrentPage() {
            return IntegerParser.parse(request.getParameter(PAGING_CURRENT));
        }
    });
    tsradingManage.export(tsradings.getItems(),
response.getOutputStream(), "");
    }
}

```

点击“导出”按钮，可以根据需要将查询到的委托记录导出为 Excel 文件，下载到本地。

## (2) 撤销及批量撤销

在某些情况下，当需要对某个用户或某个市场下的某些委托进行撤销或批量撤销时，可以选择“撤销”，或者在选中所有筛选出来的委托记录后点击“批量撤销”按钮。

撤销和批量撤销委托代码如下：

```

@Right(id = "C_TRADING_CANCELENTTRUST ", name = "委托管理-委托撤销")
public class CanceleEntrust extends AbstractTradingServlet {
    private static final long serialVersionUID = 1L;
    @Override
    protected void processGet(HttpServletRequest request, HttpServletResponse
response, ServiceSession serviceSession) throws Throwable {
        processPost(request, response, serviceSession);
    }
    @Override
    protected void processPost(HttpServletRequest request, HttpServletResponse
response, ServiceSession serviceSession) throws Throwable {
        int id = IntegerParser.parse(request.getParameter("id"));
        int scid = IntegerParser.parse(request.getParameter("scid"));
        int userid=IntegerParser.parse(request.getParameter("userid"));
        TradingManage tradingManage = serviceSession.getService
(TradingManage.class);
        tradingManage.canceleEntrust (id,scid,userid);
        sendRedirect(request, response, getController().getURI(request,
EntrustList.class));
    }
}

```

撤销委托涉及的表和判断逻辑比较多，接下来我们将分成三个部分进行详细介绍。





## ① 获取参数。

```
// 撤销
@Override
public void cancelEntrust (int id, int... parameter) throws Throwable {
    serviceResource.openTransactions();
    try {
        int userid = 0;
        if (parameter.length >= 2) {
            userid = parameter[1];
        } else {
            userid = serviceResource.getSession().getAccountId();
        }
        if (userid <= 0) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("用户不存在, userID="+userid);
            }
            throw new ParameterException("用户不存在, 请联系客服。");
        }
        if (id <= 0) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("委托不存在, id="+id);
            }
            throw new ParameterException("委托不存在。");
        }
        if (parameter[0] <= 0) {
            if (LOGGER.isWarnEnabled()) {
                LOGGER.warn("市场不存在, marketId="+parameter[0]);
            }
            throw new ParameterException("市场不存在。");
        }
        EntrustType entrustType = null; // 委托状态
        TransStatus transStatus = null; // 交易类型
        BigDecimal entrustPrice = new BigDecimal(0); // 委托价格
        BigDecimal entrustAmount = new BigDecimal(0); // 委托数量
        BigDecimal transVolume = new BigDecimal(0); // 已成交数量
        BigDecimal transMoney = new BigDecimal(0); // 成交金额

        int marketId = 0; // 市场 ID
        int partitionId = 0; // 分区 ID
        Connection connection = getConnection(P2PConst.DB_USER);
        PreparedStatement ps = connection.prepareStatement("SELECT entrustPrice,
entrustAmount, transVolume, TransStatus, transStatus, transMoney, marketId FROM
```





```
UserEntrusInfo WHERE id=? FOR UPDATE");
    ps.setInt(1, id);
    ResultSet re = ps.executeQuery();
    while (re.next()) {
        entrustPrice = re.getBigDecimal(1);
        entrustAmount = re.getBigDecimal(2);
        transVolume = re.getBigDecimal(3);
        TransStatus = EnumParser.parse(TransStatus.class, re.getString(4));
        entrustType = EnumParser.parse(EntrustType.class, re.getString(5));
        transMoney = re.getBigDecimal(6);
        marketId = re.getInt(7);
    }
    if (entrustType == null || entrustType != EntrustType.JYZ) {
        throw new ParameterException("委托已处理。");
    }
    if (TransStatus == null) {
        throw new ParameterException("委托不存在。");
    }
}
```

②锁定相关数据库表，计算买入委托应该返还的金额。

```
// 根据市场 ID 查询交易分区-买入币种 ID
buy_coinId = selectInt(P2PConst.DB_USER, "SELECT buy_coin FROM
marketInfo WHERE id=?", marketId);
// 0 为法币，大于 0 为代币
if (buy_coinId > 0) {
    if (TransStatus.BUY == TransStatus) {
        selectInt(P2PConst.DB_USER, "SELECT id FROM userTokeninfo
WHERE userId=? AND coinId=? FOR UPDATE", userid, buyCoinId); // 锁定用户表
        BigDecimal refund =
(entrustPrice.multiply(entrustAmount).subtract(transMoney)).setScale(8,
BigDecimal.ROUND_HALF_UP); // 返还金额
        if (refund.compareTo(new BigDecimal(0)) > 0) {
            execute(getConnection(P2PConst.DB_USER), "UPDATE
UserTokenInfo SET refund=freezeAmount-?, availBalance=availBalance+? WHERE
userId=? AND buyCoinId=?", refund, refund, userid, buyCoinId);
        } else {
            if (TransStatus.BUY == transStatus) {
                selectInt(P2PConst.DB_USER, "SELECT userId FROM
UserUSTInfo WHERE userId=? FOR UPDATE", userid); // 锁定用户表
                BigDecimal refund = (entrustPrice.multiply(entrustPrice).subtract
(transMoney)).setScale(8, BigDecimal.ROUND_HALF_UP); // 返还金额
                if (refund.compareTo(new BigDecimal(0)) > 0) {
```



```
execute(getConnection(P2PConst.DB_USER), "UPDATE
UserUSTInfo SET freeze=freeze-?,avaliBlance=avaliBlance+? WHERE
userId=?",refund, refund, userid);
    }
}
}
```

首先应查看所要撤销的委托是买入委托还是卖出委托,如果是买入委托,则撤销委托时返回的是交易剩余总金额。所以在撤销买入委托时,首先应查看用户买入时的付款币种ID,然后计算该用户的剩余总额,即委托单挂出去后委托总额减去已成交的金额,剩余总额就是应该返还的金额。计算完成后,再查询该币种的冻结金额及可用金额。锁定该用户的该币种的资金信息,执行更新语句,将冻结金额更新为原冻结金额减去返还金额,将可用金额更新为原可用金额加上返还金额。

③如果要撤销的是卖出委托,则继续执行以下代码。

```
if (TransStatus.SELL == TransStatus) {
    BigDecimal refund = entrusAmount.subtract(transMoney); // 返还数量
    int coinId = selectInt(P2PConst.DB_USER, "SELECT sell_coinId
FROM marketInfo WHERE id=?", parameter[0]); // 获得币种ID
    int UserTokenInfo_id = selectInt(P2PConst.DB_USER, "SELECT id
FROM UserTokenInfo WHERE userId=? AND coinId= ?", userid, coinId);
    if (UserTokenInfo_id <= 0) {
        throw new ParameterException("币不存在。");
    }
    selectInt(P2PConst.DB_USER, "SELECT id FROM UserTokenInfo WHERE
id=? FOR UPDATE", UserTokenInfo_id); // 锁定表
    if (coinId > 0) {
        execute(getConnection(P2PConst.DB_USER),
"UPDATE UserTokenInfo SET avaliBlance=avaliAmount+?,freezeAmount= freezeAmount-?
WHERE userId=? AND coinId=?", refund, refund, userid, coinId);
    }
}

execute(getConnection(P2PConst.DB_USER), "UPDATE UserEntrustInfo
SET EntrustType=? WHERE id=?", EntrustType.UNDONE, id);
String content = "委托 ID: " + id + ";委托数量: " + entrustAmount;
log(serviceResource.getSession().getAccountId(), content);
serviceResource.commit();
} catch (Exception e) {
```





```

        serviceResource.rollback();
        e.printStackTrace();
        throw new ParameterException(e.getMessage());
    }
}

```

首先计算返还数量，然后查询该用户的该币种的资金信息，并锁定该信息，接下来执行更新语句，将冻结数量更新为原冻结数量减去返还数量，将可用数量更新为原可用数量加上返还数量。

无论是撤销买入委托还是卖出委托，都需要判断撤销委托中的币种类型，若币种 ID 为 0，则该币种为平台的法币；若大于 0，则为其他数字货币。在撤销以后，需要更新的相应数据库表中不同的币种类型是不相同的。

对于撤销委托，用户可以在前台进行操作，平台管理人员通过后台页面也可以进行操作。

## 5. 成交记录

“成交记录”页面如图 4-52 所示，显示的是平台所有市场的已成交历史记录。在这个页面中显示了大量的数据，也提供了条件查询功能，比如可以通过买家账号或卖家账号来查询某个具体用户的历史交易记录，也可以通过时间、交易市场和委托类型来查看某一时间段或某一市场的成交记录，或者某种委托类型的交易记录，方便对平台的交易情况进行分类了解。

欢迎: manager 职位: 超级管理员 [个人中心] [安全设置]

2018年05月19日 星期六

后台管理系统											
概览 用户管理 交易管理 财务管理 内容管理 系统管理											
交易管理											
成交记录											
总计: 6条											
ID	买家	卖家	币种/市场	单价(元)	数量	总额	买家手续费	卖家手续费	类型	时间	备注
20	162...@163.com	162751...@163.com	数字货币(DAO)/数字货币(DAO)	285.000000	1000000	285	0.00	0.00	买入	2018-05-18 02:14:01	
19	162751...@163.com	162751...@163.com	数字货币(DAO)/数字货币(DAO)	285.000000	2500000	712.500	0.00	0.00	卖出	2018-04-28 07:12:02	
18	162751...@163.com	162751...@163.com	数字货币(DAO)/数字货币(DAO)	1500.000000	7000000	10500000	0.00	0.00	买入	2018-04-28 06:03:07	
17	162751...@163.com	162751...@163.com	数字货币(DAO)/数字货币(DAO)	152000000	12000000	1824000000	0.00	0.00	买入	2018-04-28 04:42:37	
16	162751...@163.com	162751...@163.com	数字货币(DAO)/数字货币(DAO)	166000000	2500000	415000000	0.00	0.00	卖出	2018-04-28 04:40:57	
15	162751...@163.com	162751...@163.com	数字货币(DAO)/数字货币(DAO)	3.000000	5000000	15000	0.00	0.00	买入	2018-04-27 07:02:56	

第1页/共1页

图 4-52

查询成交记录代码如下：

```

@Right(id = "C_TRADING_TradingRecordLIST", isMenu = true, name = "成交记录")
public class TradingRecordList extends AbstractTradingServlet {

```





```

        private static final long serialVersionUID = 1L;
        @Override
        protected void processPost(final HttpServletRequest request, final
        HttpServletResponse response, final ServiceSession serviceSession) throws
        Throwable {
            TradingManage tradingManage = serviceSession.getService
            (TradingManage.class);
            TransSubareaEntity[]
            transSubarea=tradingManage.transSubareaSearch(); // 查询市场
            PagingResult<TradingRecordEntity> tradingRecord = tradingManage.
            TradingRecordSearch(new TradingQuery() { // 查询交易记录
                @Override
                public String getUsername() {
                    return request.getParameter("userName");
                }
                @Override
                public TransStatus getTransStatus() {
                    return EnumParser.parse(TransStatus.class, request.getParameter
            ("TransType"));
                }
                @Override
                public int getId() {
                    return IntegerParser.parse(request.getParameter("id"));
                }
                @Override
                public Timestamp getCreateTimeStart() {
                    return TimestampParser.parse(request.getParameter("start"));
                }
                @Override
                public Timestamp getCreateTimeEnd() {
                    return TimestampParser.parse(request.getParameter("end"));
                }
                @Override
                public String getBuyer() {
                    return request.getParameter("buyer");
                }
                @Override
                public String getSeller() {
                    return request.getParameter("seller");
                }
                @Override
                public int getDisplayNum() { // 显示条数
                    return IntegerParser.parse(request.getParameter("displayNum"));
                }
            }

```



```
    }, new Paging() {
        @Override
        public int getSize() {
            return 10;
        }
        @Override
        public int getCurrentPage() {
            return IntegerParser.parse(request.getParameter(PAGING_CURRENT));
        }
    });
    request.setAttribute("tradingRecord", tradingRecord);
    request.setAttribute("transSubarea", transSubarea);
    forwardView(request, response, getClass());
}

}
```

通过上述代码可以看出，这里查询了市场列表及交易记录表，得到一个分页查询的实体类，返回到前台页面。

市场查询 SQL 如下：

```
@Override
public TransSubareaEntity[] TransSubareaSearch() throws Throwable {
    return selectAll(getConnection(P2PConst.DB_USER), new ArrayParser
<TransSubareaEntity>() {
        ArrayList<TransSubareaEntity> list = new ArrayList<TransSubareaEntity>();
        @Override
        public TransSubareaEntity[] parse(ResultSet re) throws SQLException {
            while (re.next()) {
                TransSubareaEntity j = new TransSubareaEntity();
                j.id = re.getInt(1);
                j.nameEn = re.getString(2);
                j.nameCn = re.getString(3);
                list.add(j);
            }
            return list == null ? null : list.toArray(new TransSubareaEntity
[list.size()]);
        }
    }, "SELECT MarketInfo.id id,buy.nameEn nameEn,buy.nameCn nameCn,
sell.nameEn nameEn,sell.nameCn nameCn FROM MarketInfo LEFT JOIN CoinInfo buy ON
MarketInfo.buy_coinId=sell.id LEFT JOIN CoinInfo sell ON
sell.id=MarketInfo.sell-coinId");
}
```





这里查询出来的交易市场的实体类集合，会显示在页面中的“市场”下拉菜单中，用于在查询时选择不同的市场，以查看该市场下的交易成交记录。

成交记录查询 SQL 如下：

```
@Override
public PagingResult<TradingRecordEntity> TradingRecordSearch
(TradingQuery tradingQuery, Paging paging) throws Throwable {
    StringBuilder s = new StringBuilder(
        "SELECT UserTradingRecordInfo.id ID,UserTradingRecordInfo.unitPrice
unitPrice,UserTradingRecordInfo.quantity quantity,UserTradingRecordInfo.totalAmount
totalAmount,UserTradingRecordInfo.buyer_serviceCharge buyer_serviceCharge,
UserTradingRecordInfo.seller_serviceCharge seller_serviceCharge UserTradingRecordInfo.
type type,UserTradingRecordInfo.time time, "
        + "UserTradingRecordInfo.buyerId buyer,UserTradingRecordInfo.sellerId
seller, "
        + "buyCoin.nameEn nameEn,buyCoin.nameCn nameCn, sellCoin. nameEn,
sellCoin.nameCn nameCn"
        + "FROM UserTradingRecordInfo LEFT JOIN MarketInfo ON
UserTradingRecordInfo.marketId=MarketInfo.id "
        + "LEFT JOIN UserAccount ON UserTradingRecordInfo.buyerId=
UserAccount.id"
        + " LEFT JOIN CoinInfo buyCoin ON buyCoin.id=MarketInfo. buyCoinId"
        + " LEFT JOIN CoinInfo sellCoin ON sellCoin.id=MarketInfo.sellCoinId
" + " LEFT JOIN UserAccountInfo seller_id ON seller_id.id=UserTradingRecordInfo.
sellerId "
        + " LEFT JOIN UserAccountInfo seller_id ON seller_id.id=
UserTradingRecordInfo.sellerId " + " WHERE 1=1");
    ArrayList<Object> parameters = new ArrayList<>();
    int userid = serviceResource.getSession().getAccountId();
    if (userid == 17) {
        s.append(" AND UserTradingRecordInfo.marketId=6 ");
        s.append(" ORDER BY UserTradingRecordInfo.dealTime DESC LIMIT 100");
    } else {
        if (tradingQuery != null) {
            SQLConnectionProvider sqlConnectionProvider =
getSQLConnectionProvider();
            if (tradingQuery.getId() > 0) {
                s.append(" AND UserTradingRecordInfo.marketId=?");
                parameters.add(tradingQuery.getId());
            }
            if (!StringHelper.isEmpty(tradingQuery.getBuyer())) {
                s.append(" AND buyer_id.id LIKE ?");
            }
        }
    }
}
```





```

        parameters.add(sqlConnectionProvider.allMatch
(tradingQuery.getBuyer()));
    }
    ..... // 省略部分查询条件判断
    if (tradingQuery.getCreateTimeEnd() != null) {
        s.append(" AND TIMESTAMP(UserTradingRecordInfo.dealTime) <= ?");
        parameters.add(tradingQuery.getCreateTimeEnd());
    }
}
int i = 100;
if (tradingQuery.getDisplayNum() > 0) {
    i = tradingQuery.getDisplayNum();
}
s.append(" ORDER BY UserTradingRecordInfo.dealTime DESC LIMIT ?");
parameters.add(i);
}
return selectPaging(getConnection(P2PConst.DB_USER), new ArrayParser
<TradingRecordEntity>() {
    @Override
    public TradingRecordEntity[] parse(ResultSet re) throws SQLException {
        ArrayList<TradingRecordEntity> list = new ArrayList
<TradingRecordEntity>();
        while (re.next()) {
            TradingRecordEntity jy = new TradingRecordEntity();
            jy.id = re.getInt(1);
            jy.dj = re.getBigDecimal(2);
            jy.amount = re.getBigDecimal(3);
            ..... // 省略其余赋值
            if (list == null) {
                list = new ArrayList<>();
            }
            list.add(jy);
        }
        return list == null ? null : list.toArray(new TradingRecordEntity
[list.size()]);
    }
}, paging, s.toString(), parameters);
}

```

查询出平台交易记录后，封装成分页实体类，返回到控制器中，再返回到前台页面进行页面渲染。

该列表同样支持将成交记录导出为 Excel 文件。具体代码与委托记录相同。

## 6. 转入统计

用户获取法币或代币的途径有两种：一是在“币币交易”中心，通过事先在 C2C 中充值的平台币从其他用户手中购买想拥有的币种；二是从自己的钱包中将资产转入该平台。“转入统计”页面显示的即为用户转入平台资产的转入记录，如图 4-53 所示。

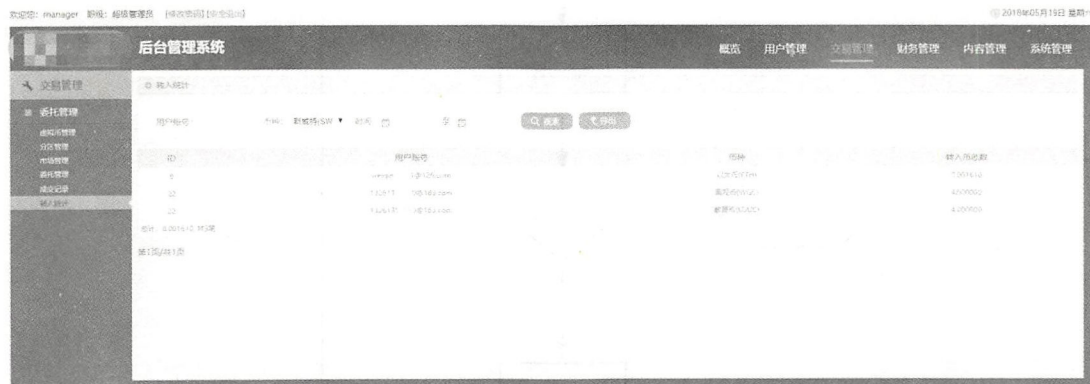


图 4-53

该页面同样支持按条件查询列表，以及将数据导出为 Excel 文件。转入记录查询的实现与成交记录查询相似，只是参数不同。

用户资产转入记录表结构如下：

```
CREATE TABLE `UserCoinInto` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '自增 ID',
  `userId` int(10) unsigned DEFAULT NULL COMMENT '用户 ID, 参考 UserAccount.id',
  `coinId` int(10) DEFAULT NULL COMMENT '币种 ID, 参考 CoinInfo.id',
  `IntoNum` decimal(20,8) DEFAULT '0.0000000' COMMENT '转入数量',
  `receiveNum` decimal(20,8) DEFAULT '0.0000000' COMMENT '实际到账数量',
  `FromAdress` varchar(50) CHARACTER SET utf8 DEFAULT '' COMMENT '转出钱包地址(从哪里来)',
  `TransTime` timestamp NULL DEFAULT NULL COMMENT '交易时间',
  `ToAdress` varchar(50) CHARACTER SET utf8 DEFAULT '' COMMENT '转入钱包地址(到哪里去)',
  `is_success` enum('S','F') CHARACTER SET utf8 DEFAULT 'F' COMMENT '是否成功',
  `remark` varchar(200) CHARACTER SET utf8 DEFAULT NULL COMMENT '备注',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=23 DEFAULT CHARSET=utf8mb4 COMMENT='虚拟转入表(UserCoinInto)';
```

通过以上介绍，我们大致了解了一个交易所添加虚拟币、添加分区、添加市场、统计数据的流程，如图 4-54 所示。

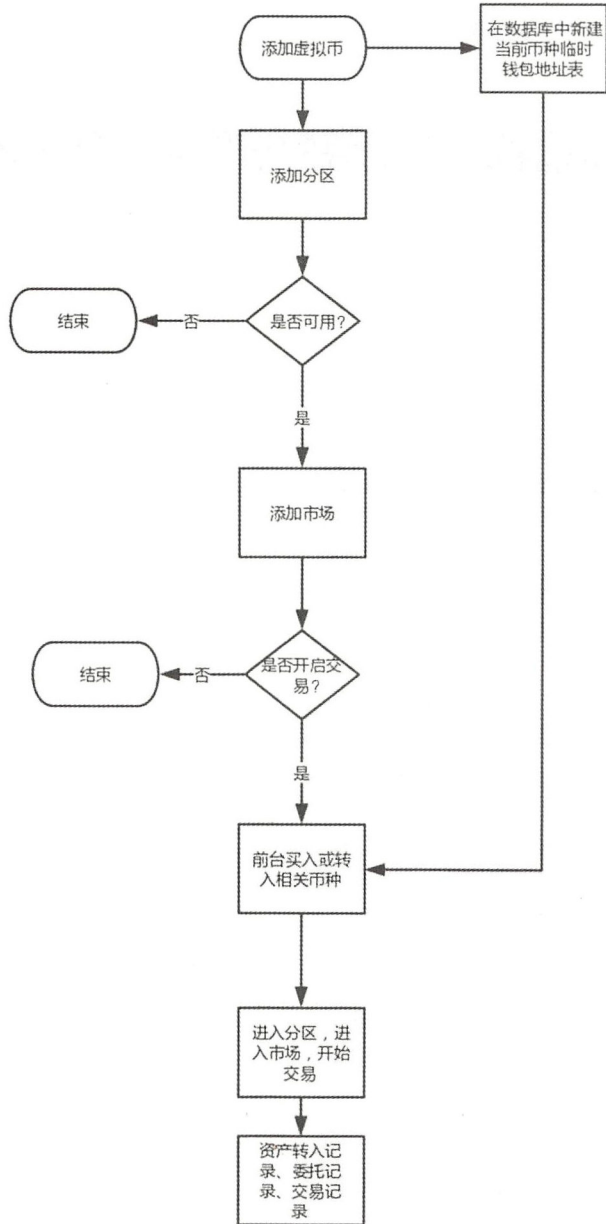


图 4-54





```

址(从哪里来)',
    `TransTime` timestamp NULL DEFAULT NULL COMMENT '交易时间',
    `TransTo` varchar(50) CHARACTER SET utf8 DEFAULT '' COMMENT '转入钱包地址
(到哪里去)',
    `is_success` enum('S','F') CHARACTER SET utf8 DEFAULT 'F' COMMENT '是否
成功',
    `remark` varchar(200) CHARACTER SET utf8 DEFAULT NULL COMMENT '备注',
    PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=23 DEFAULT CHARSET=utf8mb4 COMMENT='虚拟币
转入表(UserCoinInto)';

```

用户将虚拟币转入平台为其生成的该币种的临时钱包，之后临时钱包中的资产会自动转入热钱包，再自动转入平台冷钱包，这个过程是由一个定时任务来执行的。平台根据用户的虚拟币转入表，查询到该用户的转入金额，在其虚拟币资产表中插入或更新该虚拟币的资产信息。之后用户在平台上的交易是不涉及区块链的，只是在平台数据库中该用户的资金发生变化，其转入的真正资产已经进入平台的相应虚拟币冷钱包中被妥善保管。

## (2) 虚拟币转出

当用户想要将自己的资产从平台钱包转出到自己的其他钱包中时，可以申请虚拟币转出。前台用户申请虚拟币转出，验证完所有信息后提交申请，在后台的虚拟币转出表中就会出现相应的申请资产转出记录。虚拟币转出申请表如图 4-56 所示。该列表支持条件查询。



图 4-56

虚拟币转出申请表结构如下：

```

CREATE TABLE `UserCoinOutInfo` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '自增 ID',
  `userId` int(10) unsigned DEFAULT NULL COMMENT '用户 ID, 参考 UserAccount.id',
  `coinId` int(10) DEFAULT NULL COMMENT '币种 ID, 参考 CoinInfo.id',
  `walletId` int(10) DEFAULT NULL COMMENT '钱包 ID; 参考 UserWalletInfo.id',

```

```

`RollOutAmount` decimal(20,8) DEFAULT '0.00000000' COMMENT '转出数量',
`TransOutServiceCharge` decimal(20,8) DEFAULT '0.00000000' COMMENT '提现
手续费',
`TransTime` timestamp NULL DEFAULT NULL COMMENT '交易时间',
`is_success` enum('S','F') DEFAULT 'F' COMMENT '是否成功',
`remark` varchar(50) DEFAULT NULL COMMENT '备注',
`status` enum('ZCSB','ZCCG','WSH','SHTG','SHSB') DEFAULT 'WSH' COMMENT '
状态,WSH:未审核;SHTG:审核通过;SHSB:审核失败;ZCCG:转出成功;ZCSB:转出失败',
`verifier` int(11) DEFAULT NULL COMMENT '审核人',
`verifierTime` timestamp NULL DEFAULT NULL COMMENT '审核时间',
`RollOutPerson` int(11) DEFAULT NULL COMMENT '转出人',
`RollOutTime` timestamp NULL DEFAULT NULL COMMENT '转出时间',
`is_repleOrder` enum('S','F') DEFAULT 'F' COMMENT '是否补单',
`TrandingHash` varchar(200) DEFAULT NULL COMMENT '交易哈希',
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8 COMMENT='虚拟币转出
申请表UserOutInfo' ;

```

虚拟币转出需要进行审核。点击每条申请最后的“审核”，进入“审核详情”页面，如图 4-57 所示。当数据量很大时，可以点击“批量审核通过”或“批量审核不通过”按钮。在“审核详情”页面中主要查看用户申请的转出数量、手续费及转出地址，确认无误后，点击“审核通过”按钮；若有错误，则在“备注”中填写错误原因，然后点击“审核不通过”按钮。

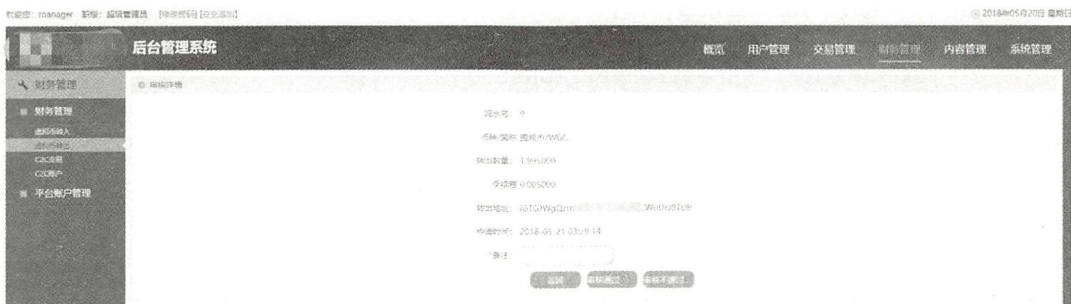


图 4-57

申请虚拟币转出审核代码如下：

```

@Override
public void updateCoinOut(int id, CoinOutStatus status, String remark)
throws Throwable {
    serviceResource.openTransactions();
}

```



```

try {
    int accountId = serviceResource.getSession().getAccountId();
    if (id <= 0) {
        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("指定的记录不存在,id="+id);
        }
        throw new ParameterException("指定的记录不存在。");
    }
    CoinOutStatus stats=EnumParser.parse(CoinOutStatus.class,
selectString (getConnection(P2PConst.DB_USER), "SELECT status FROM
UserCoinOutInfo WHERE id=? FOR UPDATE",id));
    if(stats!=CoinOutStatus.WSH){ // 未审核
        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("状态异常,stats="+stats);
        }
        throw new ParameterException("状态异常");
    }
    if (status == null) {
        if (LOGGER.isWarnEnabled()) {
            LOGGER.warn("状态不能为空,status =" + status);
        }
        throw new ParameterException("状态不能为空。");
    }
    if (status == CoinOutStatus.SHSB) { // 审核失败
        RollOutDefeat(id);
    }
    execute(getConnection(P2PConst.DB_USER),
        "UPDATE UserCoinOutInfo SET status=?,remark=?, verifier=?,
verifierTime=CURRENT_TIMESTAMP() WHERE id=?", status, remark, accountId, id);
} catch (Throwable e) {
    serviceResource.rollback();
    e.printStackTrace();
    if (LOGGER.isErrorEnabled()){
        LOGGER.error(e.getMessage(), e);
    }
    throw new LogicalException(e.getMessage());
}
}

```

通过上述代码可以看到，当审核通过后会继续向下执行，然后更新数据库信息。若审核不通过，则会调用审核失败的方法。

审核失败的方法如下:

```
/**
 * 审核不通过, 将所申请的转出资产返回到用户账户中
 */
private void RollOutDefeat(int id) throws Throwable {
    if (id <= 0) {
        throw new ParameterException("指定的记录不存在.");
    }
    int userid = 0;
    int bid = 0;
    BigDecimal amount = new BigDecimal(0); // 数量
    BigDecimal ServiceCharge = new BigDecimal(0); // 手续费
    try (Connection connection = getConnection(P2PConst.DB_USER)) {
        try (PreparedStatement ps = connection.prepareStatement("SELECT
        userId, coinId, rollOurAmount, rollOutServiceCharge FROM UserCoinOutInfo WHERE
        id=? FOR UPDATE")) { // 锁定市场表
            ps.setInt(1, id);
            try (ResultSet re = ps.executeQuery()) {
                while (re.next()) {
                    userid = re.getInt(1);
                    coinId = re.getInt(2);
                    rollOurAmount = re.getBigDecimal(3);
                    rollOutServiceCharge = re.getBigDecimal(4);
                }
            }
        }
    }

    int userCoinInfo_CoinId = selectInt(P2PConst.DB_USER, "SELECT id FROM
    UserCoinInfo WHERE userId=? AND coinId= ?", userid, coinId);
    if (UserCoinInfo_CoinId <= 0) {
        throw new ParameterException("币不存在.");
    }

    selectInt(P2PConst.DB_USER, "SELECT id FROM UserCoinInfo WHERE id=? FOR
    UPDATE", userCoinInfo_CoinId); // 锁定表
    execute(getConnection(P2PConst.DB_USER), "UPDATE UserCoinInfo SET
    useableAmount=useableAmount+?, freezeAmount=freezeAmount-? WHERE userId=? AND
    coinId=?", amount.add(serviceCharge), amount.add(serviceCharge), userid, coinId);
    // 添加提币失败的站内信息
    String rollOut = selectString(P2PConst.DB_USER, "SELECT nameEn FROM
    CoinInfo WHERE id=?", coinId);
```

```

smsdefeat(userid, amount.add(serviceCharge), rollOut);
}

```

如果审核不通过，则会将用户所申请的转出资产返回到用户账户中，并显示一条转出资产失败的站内信息。

如果审核通过，则会进入审核通过列表页面，如图 4-58 所示。



图 4-58

查询已审核的转出申请，点击每条申请最后的“转出审核”，进入最后的审核详情页面，如图 4-59 所示。

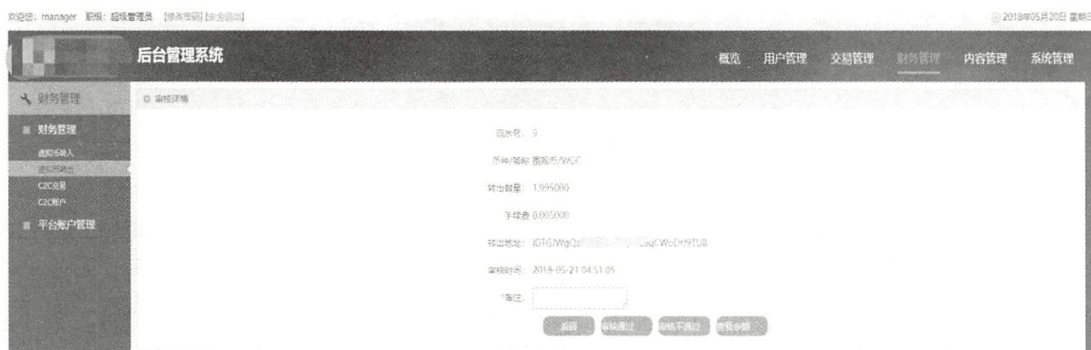


图 4-59

平台财务人员在页面中做最后的确认工作，此时首先应该查看平台的该币种的提币钱包中是否有足够的金额供转出。点击“查看余额”按钮，进入平台虚拟币账户钱包信息页面，如图 4-60 所示。



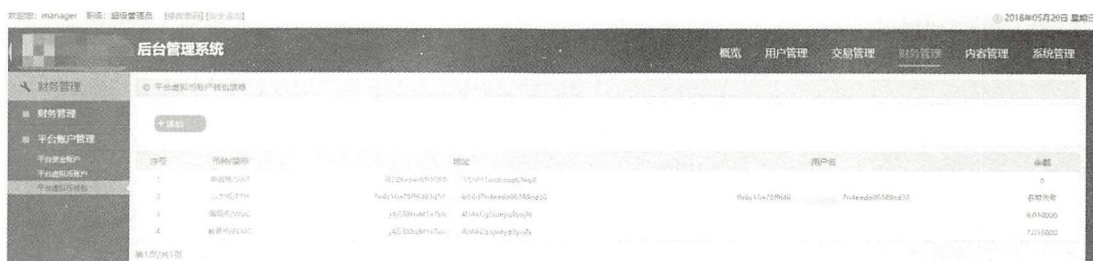


图 4-60

确认好该币种的提币钱包中的余额足够这次转出后，就可以点击“审核通过”按钮，系统会自动从平台的提币钱包向用户提币账户中转入相应的资产。

转出成功的方法如下：

```
@Override
public void updateTransCoinAudit(int id, RollOutStatus status, String remark)
throws Throwable {
    try {
        int accountId = serviceResource.getSession().getAccountId();
        if (id <= 0) {
            throw new ParameterException("指定的记录不存在。");
        }
        if (status == null) {
            throw new ParameterException("状态不能为空。");
        }
        CoinOutStatus stats = EnumParser.parse(CoinOutStatus.class, selectString
(getConnection(P2PConst.DB_USER), "SELECT status FROM UserCoinOutInfo WHERE id=?
FOR UPDATE", id)); // 锁定提币表
        if (stats != CoinOutStatus.SHTG) {
            throw new ParameterException("状态异常");
        }
        if (status == CoinOutStatus.ZCCG) {
            int userid = 0;
            int coinId = 0;
            BigDecimal amount = new BigDecimal(0);
            BigDecimal serviceCharge = new BigDecimal(0);
            IsPass is = null; // 状态
            try (Connection connection = getConnection(P2PConst.DB_USER)) {
                try (PreparedStatement ps = connection.prepareStatement
("SELECT userId, coinId, amount, serviceCharge, is_success FROM UerCoinOutInfo
```

## 区块链：交易系统开发指南

```

WHERE id=? ") {
    ps.setInt(1, id);
    try (ResultSet re = ps.executeQuery()) {
        while (re.next()) {
            userid = re.getInt(1);
            coinId = re.getInt(2);
            amount = re.getBigDecimal(3);
            serviceCharge = re.getBigDecimal(4);
            Is_success = EnumParser.parse(IsPass.class,
re.getString(5));
        }
    }
}

int UserCoinInfo_coinId = selectInt(P2PConst.DB_USER, "SELECT id
FROM UserCoinInfo WHERE userId=? AND coinId= ?", userid, coinId);
if (UserCoinInfo_coinId <= 0) {
    throw new ParameterException("币不存在。");
}
if (is_success == null || is_success == IsPass.S) {
    throw new ParameterException("已转出。");
}
// 加行锁，并查询余额
BigDecimal balance = selectBigDecimal(P2PConst.DB_USER, "SELECT
useableAmount + freezeAmount FROM UserCoinInfo WHERE id=? FOR UPDATE",
UserCoinInfo_coinId); // 锁定表
execute(getConnection(P2PConst.DB_USER), "UPDATE UserCoinInfo
SET freezeAmount=freezeAmount-? WHERE userId=? AND coinId=?",
amount.add(serviceCharge), userid, coinId);
String s = "提币手续费:" + serviceCharge;
int UserCoinTransInfo = insert(getConnection(P2PConst.DB_USER),
"INSERT INTO UserCoinTransInfo SET userId=?, coinId=?, TransType=?,
TransTime=CURRENT_TIMESTAMP(), TransAmount=?, relevanceId=?, remark=?, beforeTra
nsBalance=?, afterTransBalance=?", userid, coinId, CoinType.TBSXF, serviceCharge,
id, s, balance, balance.subtract(serviceCharge));
s = "转出数量: " + amount;
execute(getConnection(P2PConst.DB_USER),
"INSERT INTO UserCoinTransInfo SET userId=?, coinId=?, TransType=?,
TransTime=CURRENT_TIMESTAMP(), TransAmount=?, relevanceId=?, remark=?, beforeTra
nsBalance=?, afterTransBalance=?", userid, coinId, CoinType.TBSXF, serviceCharge,

```



```

id, s, balance, balance.subtract(serviceCharge));
        BigDecimal platFormBalance = selectBigDecimal(P2PConst.DB_CONSOLE,
"SELECT balance FROM PlatFormBalance WHERE id=? FOR UPDATE", coinId); // 锁定表
        execute(getConnection(P2PConst.DB_CONSOLE),
        "INSERT INTO PlatFormBalance SET id=?,totalAmount=?,totalIncome=?"
+ " ON DUPLICATE KEY UPDATE totalAmount=totalAmount+?,totalIncome=totalIncome+?",
coinId,serviceCharge, serviceCharge, serviceCharge, serviceCharge);
        execute(getConnection(P2PConst.DB_CONSOLE),
        "INSERT INTO PlatFormCoinTransInfo SET coinId=?,TransType=?,
TransTime=CURRENT_TIMESTAMP(),income=?,expend=0,balance=?,reletionId=?,remark
=?,userId=?", coinId, PlatFormCoinType.TBSXF, serviceChage, platFormBalance.add
(serviceChage), PlatFormCoinTransInfo_id, remark, userid);
        execute(getConnection(P2PConst.DB_USER), "UPDATE UserCoinRollOutInfo
SET is_success=? WHERE id=?", IsPass.S, id);
        // 添加提币成功的站内信息
        String rollOutSuccess = selectString(P2PConst.DB_USER, "SELECT
nameEn FROM CoinInfo WHERE id=?", coinId);
        smssuccess(userid, amount.add(service), rollOutSuccess);
    } else {
        RollOutDefeat(id);
    }
    execute(getConnection(P2PConst.DB_USER),
    "UPDATE UserCoinRollOutInfo SET status=?,remark=?, TransOuter=?,
rollOutTime=CURRENT_TIMESTAMP() WHERE id=?", status, remark, accountId, id);
} catch (IOException e) {
    serviceResource.rollback();
    e.printStackTrace();
    throw new LogicalException(e.getMessage());
}
}

```

审核通过后, 该条记录会被转移到转出成功列表中, 如图 4-61 所示; 审核不通过, 则会被转移到审核未通过列表中; 而在转出过程中失败时, 则会被转移到转出失败列表中。这三个列表的查询内容相同, 不同的是它们的查询条件, 以及在查询时所使用的用户虚拟币转出信息表中的转出状态。



[illegible]

图 4-61

在转出成功列表的“操作”下有一个“补单”选项，这是为用户反映自己没有收到转出的资产，平台核实确实没有转出成功时使用的功能。点击“补单”，会从平台的提币钱包中再次给用户的提币账户转入用户申请的转出资产。

补单代码如下:

```
@Override
public void RollOutReplenishments(int id) throws Throwable {
    serviceResource.openTransactions();
    if (id <= 0) {
        throw new ParameterException("指定的记录不存在。");
    }
    try {
        int accountId = serviceResource.getSession().getAccountId();
        int userid = 0;
        int coinId = 0;
        BigDecimal amount = new BigDecimal(0);
        BigDecimal serviceCharge = new BigDecimal(0);
        IsPass is_success = null; // 状态: 是否成功
        RollOutCoinStatus status = null; // 状态
        IsPass is_Replenishments = null; // 是否补过单
        try (Connection connection = getConnection(P2PConst.DB_USER)) {
            try (PreparedStatement ps = connection.prepareStatement("SELECT
userId,coinId,TransAmount,RollOutServiceCharge,is_success,status,is_Replenis
hments FROM UserCoinOutInfo WHERE id=? FOR UPDATE")) { // 锁定市场表
```

```

        ps.setInt(1, id);
        try (ResultSet re = ps.executeQuery()) {
            while (re.next()) {
                userid = re.getInt(1);
                coinId = re.getInt(2);
                amount = re.getBigDecimal(3);
                serviceCharge = re.getBigDecimal(4);
                Is_success = EnumParser.parse(IsPass.class, re.getString(5));
                status = EnumParser.parse(RollOutCoinStatus.class,
re.getString(6));
                Is_Replenishments=EnumParser.parse(IsPass.class,re.getString(7));
            }
        }
    }

    int userCoinTransInfo_coinId = selectInt(P2PConst.DB_USER, "SELECT
id FROM UserCoinInfo WHERE F02=? AND F03= ?", userid, bid);

    if (userCoinTransInfo_coinId <= 0) {
        throw new ParameterException("币不存在。");
    }
    if (is_success == null || is_success != IsPass.S) {
        throw new ParameterException("未转出。");
    }
    if (status == null || status != RollOutCoinStatus.ZCCG) {
        throw new ParameterException("未转出成功。");
    }
    If (is_Replenishments==null||Replenishments==IsPass.S){
        throw new ParameterException("已经补过单。");
    }
    selectInt(P2PConst.DB_USER, "SELECT id FROM UserCoinTransInfo WHERE
id=? FOR UPDATE", UserCoinTransInfo_coinId);// 锁定表
    execute(getConnection(P2PConst.DB_USER), "UPDATE UserCoinInfo SET
usebleAmount=usebleAmount+? WHERE userId=? AND coinId=?", amount.add
(serviceCharge), userid, bid);
    String replenishments = "补单数量: " + amount.add(serviceCharge);
    int userCoinTransInfo_coinId = insert(getConnection(P2PConst.DB_USER),

```



```

        "INSERT INTO UserCoinTransInfo SET userId=?,coinId=?,TransType=?,
TransTime=CURRENT_TIMESTAMP(),TransAmount=?,remark=?,relationId=?", userid,
coinId,CoinType.BD, amount.add(serviceCharge), id, remark);

        BigDecimal platFormBalance = selectBigDecimal(P2PConst.DB_CONSOLE,
"SELECT balance FROM PlatFormCoinBalance WHERE id=? FOR UPDATE", coinId);// 锁定表
        execute(getConnection(P2PConst.DB_CONSOLE), "UPDATE PlatCoinBalance SET
totalAmount=totalAmount-?,totalIncome=totalIncome+? WHERE id=?", serviceCharge,
serviceCharge,coinId);

        replenishments = "补单返还手续费: " + serviceCharge;
        execute(getConnection(P2PConst.DB_CONSOLE),
        "INSERT INTO PlatFormTransInfo SET bid=?,TradeType=?, TradeTime=
CURRENT_TIMESTAMP(), income=0, disbursement=?, balance=?, link_id=?, remark=?,
userid=?", bid, PtXlbType.BDFHSXF, serviceCharge, zzc.subtract(serviceCharge),
UserCoinOutInfo_id, remark, userid);
        execute(getConnection(P2PConst.DB_USER), "UPDATE UerCoinOutInfo SET
is_success=? WHERE id=?", IsPass.S, id);
        String content = "补单数量: " + amount.add(serviceCharge) + "币种 ID:
" + coinId + "订单 ID" + id;
        log(accountId, content);
    } catch (Throwable e) {
        serviceResource.rollback();
        e.printStackTrace();
        throw new LogicalException(e.getMessage());
    }
}

```

首先查看用户虚拟币转出表，看该转出记录的状态是否为转出成功。若转出成功，则直接返回；若没有转出成功，则首先将用户减去的资产返还，然后重新执行转出程序，更新平台相应虚拟币的账户余额。

虚拟币转出涉及的步骤比较多，使用流程图来表达可能会更直观些。如图 4-62 所示为井通系虚拟币转出流程图，如图 4-63 所示为非井通系虚拟币转出流程图。





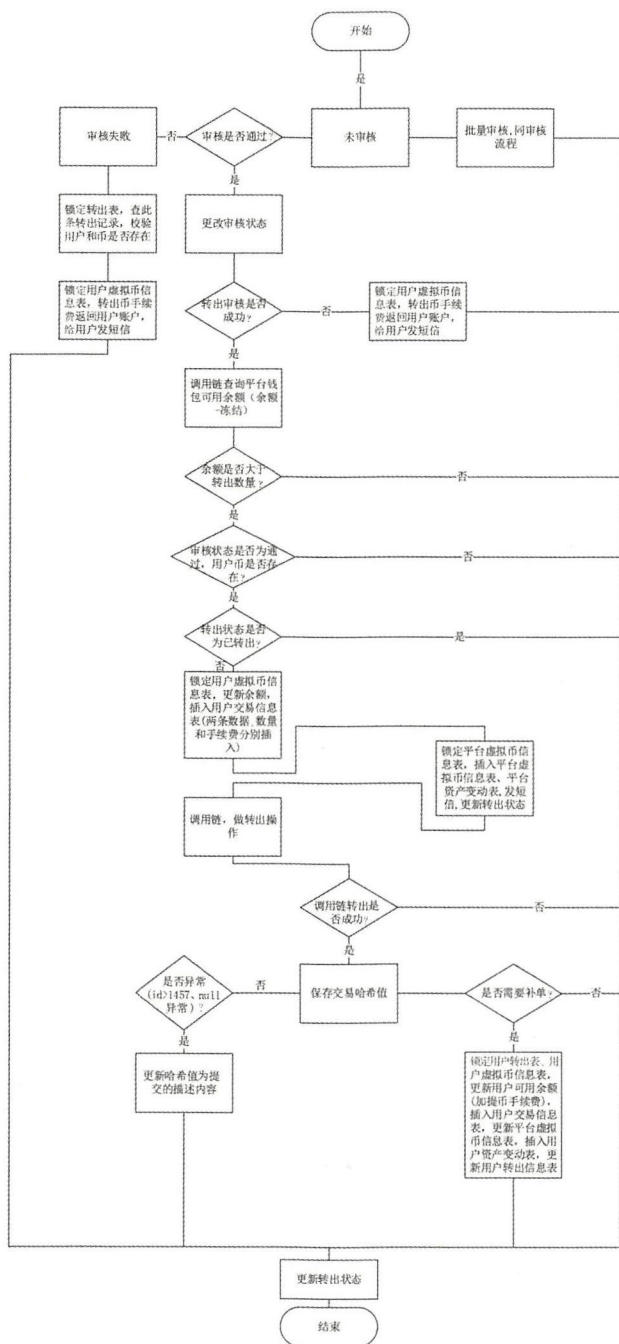


图 4-62



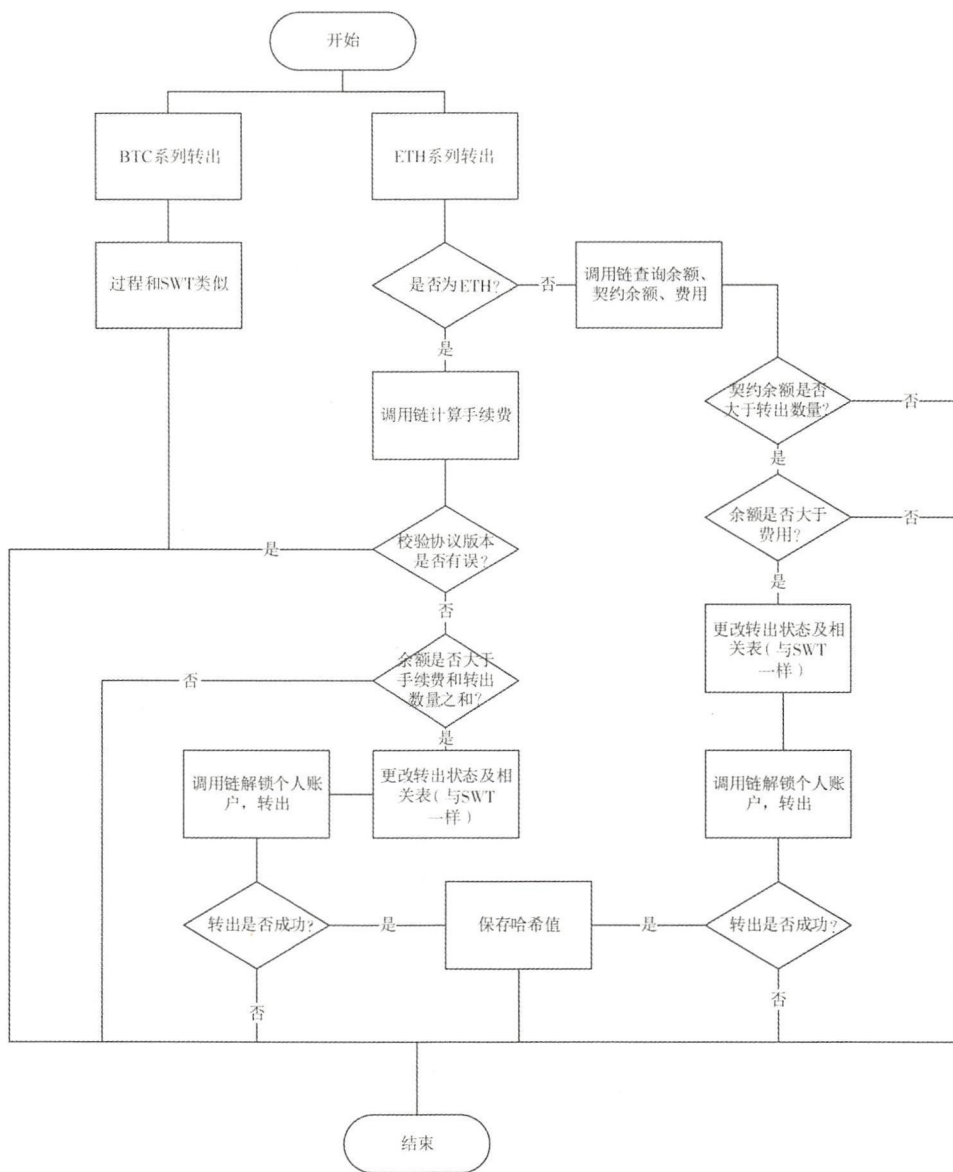


图 4-63

### (3) C2C 交易

C2C 交易即前台页面中的场外交易，这里是显示用户买入或卖出平台法币 UST 订单的地方，如图 4-64 所示。这个页面中的数据较多，支持条件查询，在“交易方式”下拉



列表框中有“买入”和“卖出”供选择，在“状态”下拉列表框中有“处理中”“未打款”“交易完成”和“撤销卖出”供选择。也支持导出为 Excel 文件，这里不再赘述。

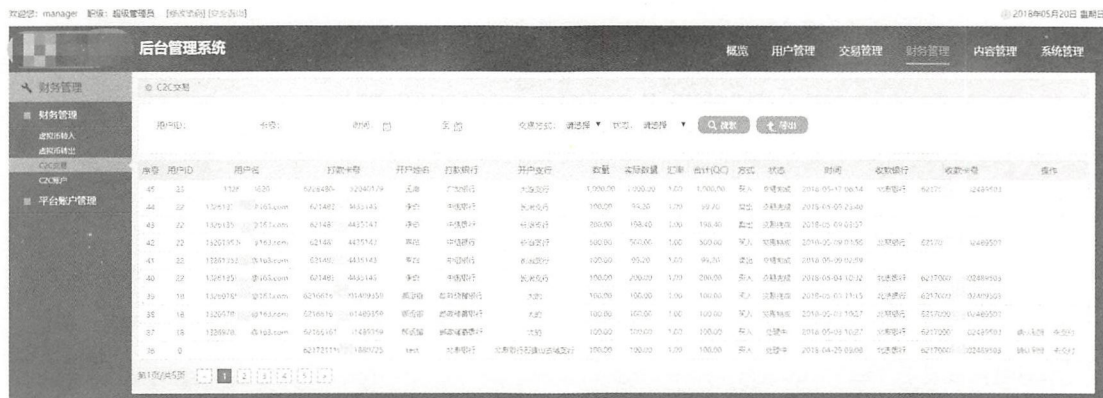


图 4-64

在前台买入或卖出订单提交后，会在后台页面中显示该订单。

C2C 交易列表查询代码如下：

```
public PagingResult<C2cEntity> getC2cList(C2cQuery query, Paging paging)
throws Throwable {
    StringBuilder sql = new StringBuilder(
        "SELECT UserAccountInfo.id userid,UserAccountInfo.username
        username,UserInfo.name name,");
    sql.append("C2CTransInfo.id id,C2CTransInfo.createTime createTime,
    C2CTransInfo. ActualToAccount,C2CTransInfo.status status,C2CTransInfo.submitAmount
    submitAmount,C2CTransInfo.exchangeRate exchangeRate,C2CTransInfo.type type,");
    sql.append("UserBankCardInfo.bankCardNum bankCardNum,BankInfo.bankOfDeposit
    bankOfDeposit,UserBankCardInfo.bankAccountNo bankAccountNo ");
    sql.append(" C2CBankInfo.bankName bankName,C2CBankInfo.bankNum bankNum,
    UserBankCardInfo.accountOwner accountOwner");
    sql.append(" FROM C2CTransInfo LEFT JOIN UserAccountInfo ON C2CTransInfo.
    userId=UserAccountInfo.id LEFT JOIN UserInfo ON UserAccountInfo.id=UserInfo.id");
    sql.append(" LEFT JOIN " + P2PConst.DB_CONSOLE + ".C2CBankInfo ON
    C2CBankInfo.id=C2CTransInfo.bankId ");
    sql.append(" LEFT JOIN UserBankCardInfo ON C2CTransInfo.bankId=
    UserBankCardInfo.id LEFT JOIN BankInfo ON UserBankCardInfo.bankId =BankInfo.id
    WHERE 1=1");
    ArrayList<Object> parameters = new ArrayList<>();
```





```

        if (query != null) {
            SQLConnectionProvider sqlConnectionProvider = getSQLConnectionProvider();
            ..... // 省略部分查询参数
            String name = query.getName();
            if (!StringHelper.isEmpty(name)) {
                sql.append(" AND UserInfo.name LIKE ?");
                parameters.add(sqlConnectionProvider.allMatch(name));
            }
        }
        sql.append(" ORDER BY C2CTransInfo.createTime DESC");
        return selectPaging(getConnection(P2PConst.DB_USER), new ArrayParser
<C2cEntity>() {
            ArrayList<C2cEntity> list = new ArrayList<C2cEntity>();

            public C2cEntity[] parse(ResultSet rst) throws SQLException {
                while (rst.next()) {
                    C2cEntity user = new C2cEntity();
                    user.userid = rst.getInt(1);
                    user.loginName = rst.getString(2);
                    user.userName = rst.getString(3);
                    ..... // 省略部分查询字段
                    user.accountOwner = rst.getString(16);
                    list.add(user);
                }
                return list.toArray(new C2cEntity[list.size()]);
            }
        }, paging, sql.toString(), parameters);
    }
}

```

在 C2C 交易买入订单后的“操作”下有“确认到账”和“未支付”两个选项。确认到账是指用户将买 UST 的钱线下充入了平台预设的收款银行卡中，当平台财务人员收到银行卡入款的信息，并确认是该用户的转账且金额正确后，点击“确认到账”，将该用户买入的 UST 充入其账户中。若实际到账金额不足，则可以按实际到账金额折算为 UST 数量，充入其账户中。如果在 24 小时内未打款，则点击“未支付”，该订单将被取消。

确认到账代码如下：

```

@Override
public void czQrdz_xlb(int id, BigDecimal sjdzje) throws Throwable {
    serviceResource.openTransactions();
}

```



```

try {
    int status = selectInt(getConnection(P2PConst.DB_USER), "SELECT
status FROM C2CTransInfo WHERE id=?", id); // 状态
    int type = selectInt(getConnection(P2PConst.DB_USER), "SELECT type
FROM C2CTransInfo WHERE id=?", id); // 是否买入
    if (C2cStatus.CLZ.getType() != status) {
        throw new ParameterException("状态不是处理中。");
    }
    if (C2cType.MR.getType() != type) {
        throw new ParameterException("类型不是买入。");
    }
    String coinId = selectString(getConnection(P2PConst.DB_USER),
"SELECT F03 FROM C2CTransInfo WHERE id=?", id);
    if (StringHelper.isEmpty(coinId) || Integer.parseInt(coinId) < 0) {
        throw new ParameterException("币不存在。");
    }
    String CoinInfo_id = selectString(getConnection(P2PConst.DB_USER),
"SELECT id FROM CoinInfo WHERE id=?", coinId);
    if (StringHelper.isEmpty(CoinInfo_id) || Integer.parseInt(CoinInfo_id) < 0) {
        throw new ParameterException("币不存在。");
    }
    if (Integer.parseInt(coinId) == 0) {
        String remark = "买入:" + actulToAccount + "元";
        BigDecimal zhye = selectBigDecimal(P2PConst.DB_USER, "SELECT
balance FROM UserUSTInfo WHERE userId=? FOR UPDATE", userId); // 用户余额

        execute(getConnection(P2PConst.DB_USER), "INSERT INTO UserUSTInfo SET
userId=?, balance=balance+?, usebleAmount=usebleAmount+? ON DUPLICATE KEY UPDATE
balance=balance+?, usebleAmount=usebleAmount+?", userId, actulToAccount,
actulToAccount, actulToAccount, actulToAccount);

        execute(getConnection(P2PConst.DB_USER), "INSERT INTO
UserUSTReChargeInfo SET userId=?, type=?, createTime=CURRENT_TIMESTAMP(), income=?,
payOut=0, balance=?, referenceId=?, is_sendMsg=?", userId, TradingType.C2CMR,
actulToAccount, balance.add(actulToAccount), coinId, remark);

        execute(getConnection(P2PConst.DB_USER), "UPDATE C2CTransInfo
SET actulToAccount=?, time=CURRENT_TIMESTAMP(), status=?, handler=? WHERE id=?",
actulToAccount, C2cStatus.JYWC.getType(),
serviceResource.getSession().getAccountId(), coinId);

```





```

        // 添加买入成功的站内信息
        smsBuySuccess(userid, actulToAccount);
    } catch (Throwable e) {
        serviceResource.rollback();
        e.printStackTrace();
        throw new LogicalException(e.getMessage());
    }
}
}

```

点击“未支付”后，执行更新 SQL 语句将用户 C2C 交易信息表中该条买入 UST 记录的状态修改为“WDK”，这次买入申请就结束了。

卖出 UST 申请的操作有一个“已打款”选项，当平台确认用户卖出信息，并已经给用户指定的收款银行卡转账后，可以点击“已打款”，执行卖出 UST 的更新操作。

卖出“已打款”代码如下：

```

@Override
public void c2c_YDK(int id) throws Throwable {
    serviceResource.openTransactions();
    try {
        if (id <= 0) {
            throw new ParameterException("ID 异常。");
        }
        int userid = selectInt(getConnection(P2PConst.DB_USER), "SELECT
userAccountId FROM C2CTransInfo WHERE id=? FOR UPDATE", id);
        if (userid <= 0) {
            throw new ParameterException("用户不存在。");
        }
        int status = selectInt(getConnection(P2PConst.DB_USER), "SELECT
status FROM C2CTransInfo WHERE id=? ", id); // 状态

        int type = selectInt(getConnection(P2PConst.DB_USER), "SELECT
type FROM C2CTransInfo WHERE id=?", id); // 是否买入
        if (C2cStatus.CLZ.getType() != status) {
            throw new ParameterException("状态不是处理中。");
        }
        if (C2cType.MC.getType() != type) {
            throw new ParameterException("类型不是卖出。");
        }
    }
}

```





```

    }
    String bid = selectString(getConnection(P2PConst.DB_USER),
"SELECT coinId FROM C2CTransInfo WHERE id=?", id);
    if (StringHelper.isEmpty(bid) || Integer.parseInt(bid) < 0) {
        throw new ParameterException("币不存在。");
    }
    BigDecimal submitAmount = selectBigDecimal(getConnection
(P2PConst.DB_USER),"SELECT submitAmount FROM C2CTransInfo WHERE id=? ",id);
    String remark= "卖出:" + submitAmount;
    if (Integer.parseInt(bid) == 0) {
        BigDecimal balance = selectBigDecimal(P2PConst.DB_USER,
"SELECT balance FROM UserUSTInfoWHERE id=? FOR UPDATE",userid);
        ConfigureProvider configureProvider = serviceResource.getResource
(ConfigureProvider.class);
        BigDecimal sellServiceCharge = new BigDecimal(configureProvider.
getProperty(SystemVariable.WITHDRAW_FV)); // 卖出手续费率

        execute(getConnection(P2PConst.DB_USER), "UPDATE UserUSTInfo
SET balance=balance-?,useableAmount=useableAmount-? WHERE id=?", submitAmount,
submitAmount, userid);

        // 添加用户资金交易记录
        String UserUSTInfo = "INSERT INTO UserUSTInfo SET userId=?,
type=?,time=CURRENT_TIMESTAMP(),income=?,payOut=?,balance=?,id=?,remark=?";

        execute(getConnection(P2PConst.DB_USER), UserUSTInfo, userid,
TradingType.C2CMC, 0, submitAmount,balance.subtract(submitAmount), id, remark);

        execute(getConnection(P2PConst.DB_USER), UserUSTInfo, userid,
TradingType.C2CMCSXF, 0, submitAmount.multiply(sellServiceCharge), balance.subtract
(submitAmount), id, TradingType.C2CMCSXF.getName());
    } // 查询平台资金余额
    String selectPlatFormAmount = "SELECT id FROM PlatFormAmount FOR UPDATE";
    BigDecimal sysBalance = selectBigDecimal(P2PConst.DB_CONSOLE,
selectPlatFormAmount);
    // 更新平台资金账户
    String updatePlatFormAmount = "UPDATE PlatFormAmount SET id=id+?,
totolBalance=totolBalance+? ";

```



```

        execute(getConnection(P2PConst.DB_CONSOLE), updatePlatformAmount,
submitAmount.multiply(sellServiceCharge), submitAmount.multiply(sellServiceCh
arge));

        // 添加平台资金交易记录
        String PlatFundtransInfo = "INSERT INTO PlatFundtransInfo SET
TransType=?,time=CURRENT_TIMESTAMP(),income=?,payOut=?,balance=?,referenceId
=?,remark=?,userAccountId=?";

        execute(getConnection(P2PConst.DB_CONSOLE), PlatFundtransInfo,
PlatformFundType.C2CMCSXF, submitAmount.multiply(sellServiceCharge), 0,
sysBalance.add(submitAmount.multiply(sellServiceCharge)), id, "C2C 卖出扣除手续费
",userid);

        // 添加卖出成功的站内信息
        smscg(userid, submitAmount.subtract(submitAmount.multiply
(sellServiceCharge)));
    }
    ConfigureProvider configureProvider = serviceResource.getResource
(ConfigureProvider.class);
    BigDecimal sellServiceCharge = new BigDecimal(configureProvider.
getProperty (SystemVariable.WITHDRAW_FV));
    execute(getConnection(P2PConst.DB_USER), "UPDATE C2CTransInfo SET
actulToAccount=?, time=CURRENT_TIMESTAMP(),status=?,handler=? WHERE id=? ",
submitAmount.subtract
(submitAmount.multiply(sellServiceCharge)), C2cStatus.JYWC.getType(),
serviceResource.getSession().getAccountId(), id);
    } catch (Throwable e) {
        serviceResource.rollback();
        e.printStackTrace();
        throw new LogicalException(e.getMessage());
    }
}
}

```

确认已打款后，程序会自动更新用户的 UST 账户余额及冻结金额，并在 C2C 交易信息表中添加该条卖出记录。在平台的资金收入表中也将添加一笔收入详情，以及这笔收入的交易详情。

C2C 交易买入或卖出流程图如图 4-65 所示。



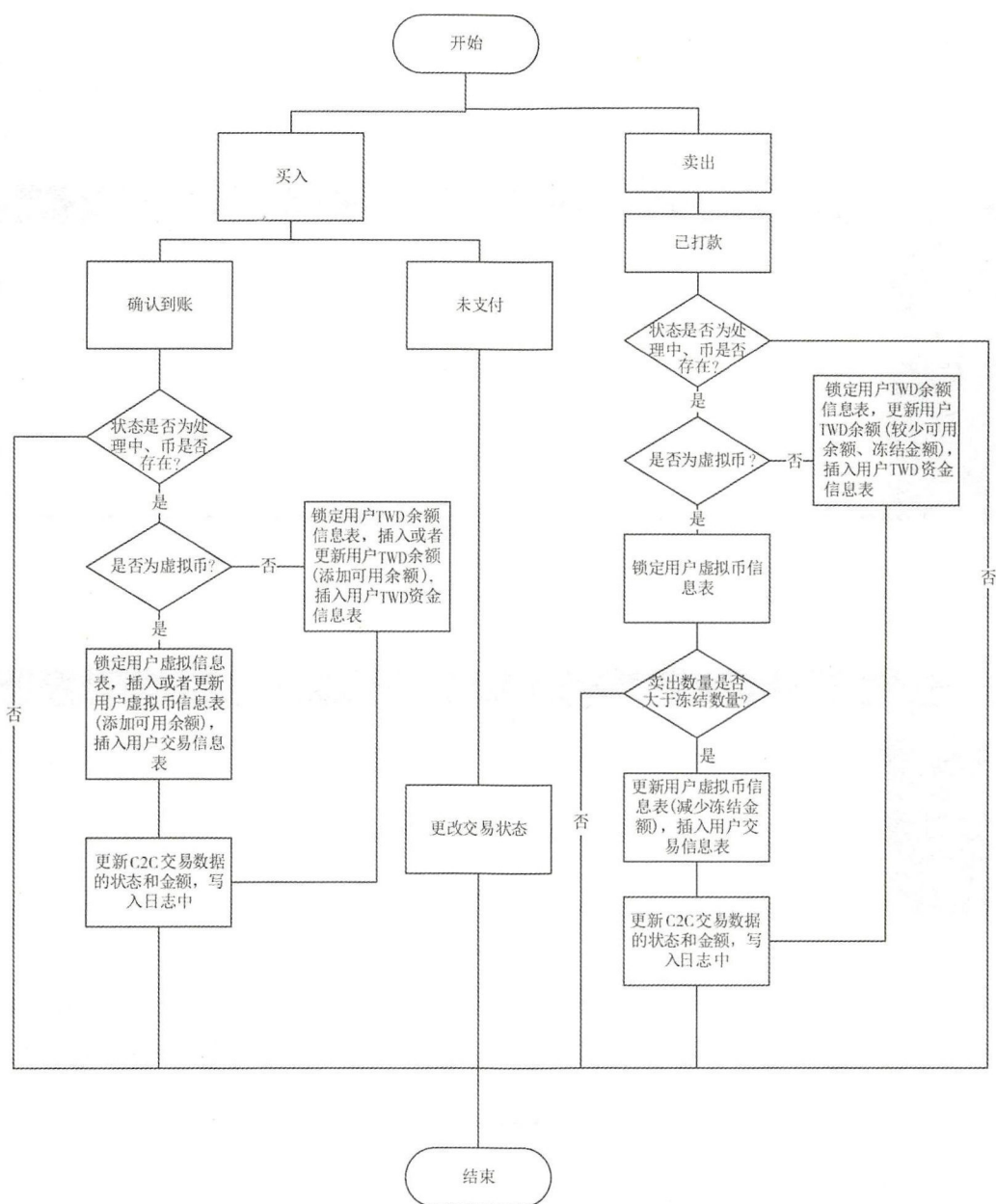


图 4-65





#### (4) C2C 账户

用户在 C2C 界面充值 UST，平台会提供多个银行卡号供用户充值。当用户在前台买入 UST 后，点击“确定”时弹出框中的银行卡号就是在这里设置的，如图 4-66 所示。

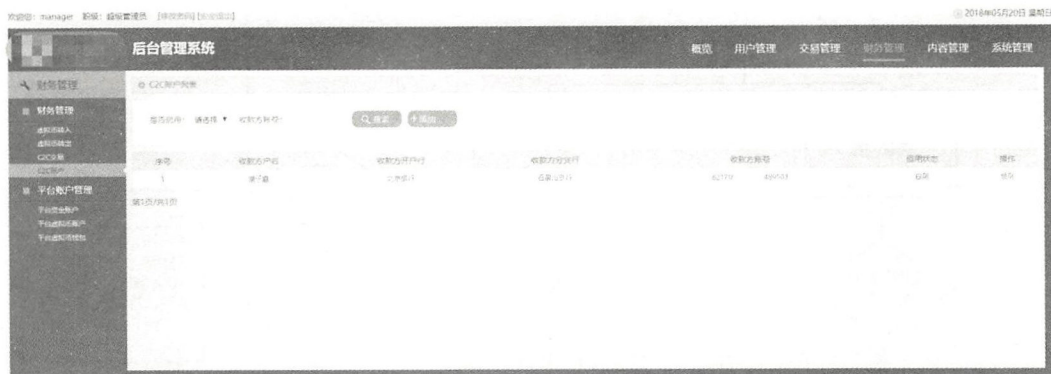


图 4-66

添加 C2C 账户页面如图 4-67 所示。

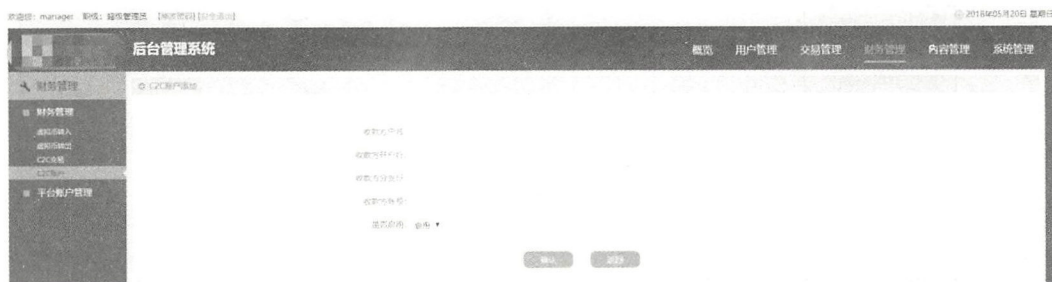


图 4-67

添加 C2C 账户需要填写收款方的户名、开户行、分支行、银行卡号，以及设置是否启用该银行卡。银行卡的启用与禁用也可以在银行卡列表页面中通过点击“启用”或“禁用”进行设置。

平台 C2C 银行卡信息表结构如下：

```
CREATE TABLE `C2CBankInfo` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '主键，自增 id',
  `cardOwner` varchar(50) DEFAULT NULL COMMENT '收款方户名',
  `bankOfDeposit` varchar(50) DEFAULT NULL COMMENT '收款方开户行',
  `bankCard` varchar(50) DEFAULT NULL COMMENT '收款方账号',
```



```

`is_use` enum('S','F') DEFAULT 'S' COMMENT '启用状态',
`constructionBank` varchar(50) DEFAULT NULL COMMENT '收款方分支',
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8 COMMENT='C2C 银行卡列表';

```

## 2. 平台账户管理

平台的资金管理一定离不开各种账户。本节我们将介绍平台各种账户的功能及创建过程。

### (1) 平台资金账户

平台资金账户储存的是用户交易及转出资产时所收取的手续费。用户每次卖出 UST 时，除了更新用户 UST 账户余额，也会更新平台资金账户余额。收取手续费是一个交易所的主要利润所在，所以通过查看平台资金账户的资金信息，可以掌握整个平台的资金情况，如图 4-68 所示。

序号	账号	用户ID	账户类型	收入	支出	结余	备注
1	2018-05-14-7014401	1355121	手续费	0.00	0.00	10.40	28
2	2018-05-11-2048112	1355121	C2C卖出手续费	0.00	0.00	10.40	44
3	2018-05-09-9131156	1355121	C2C卖出手续费	1.00	0.00	9.40	41
4	2018-05-09-0109155	1355121	C2C卖出手续费	0.80	0.00	8.60	43

图 4-68

平台资金列表支持条件查询，可以只查看交易手续费或 C2C 卖出手续费的收取情况，或者某个用户的手续费收取情况。

平台资金账户表结构如下：

```

CREATE TABLE `PlatformAmount` (
  `balance` decimal(20,8) DEFAULT '0.00000000' COMMENT '余额',
  `totalIncome` decimal(20,8) DEFAULT '0.00000000' COMMENT '总收入',
  `totalPayOut` decimal(20,8) DEFAULT '0.00000000' COMMENT '总支出'
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='平台资金账户(PlatformAmount)';

```



## (2) 平台虚拟币账户

平台虚拟币账户储存的是在用户币币交易及虚拟币转出过程中平台所收取的虚拟币手续费。一般每个币种均有一个账户。通过查看平台虚拟币账户信息，可以清楚地看到平台每个币种的手续费收入详情，如图 4-69 所示。

序号	币种	账户总额	收入总额	支出总额	操作
1	比特币 BTC	3.052360	1.054750	0.000000	查看记录
2	比特币 BNB	0.000000	0.000000	0.000000	查看记录
3	以太坊 ETH	0.000000	0.000000	0.000000	查看记录
4	莱特币 LTC	0.000000	0.000000	0.000000	查看记录
5	比特币 BCC	0.000000	0.000000	0.000000	查看记录

图 4-69

平台虚拟币账户表结构如下：

```
CREATE TABLE `PlatformTokenAccountInfo` (
  `coinId` int(11) NOT NULL DEFAULT '0' COMMENT '虚拟币 ID, 参考 CoinInfo.F01',
  `totalAmount` decimal(20,8) DEFAULT '0.00000000' COMMENT '总资产',
  `totalIncome` decimal(20,8) DEFAULT '0.00000000' COMMENT '总收入',
  `totalPayOut` decimal(20,8) DEFAULT '0.00000000' COMMENT '总支出',
  PRIMARY KEY (`coinId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='平台虚拟币账户 (PlatformTokenAccountInfo) ';
```

点击币种的账户“操作”下的“交易记录”，可以查看该币种所有手续费的交易记录，如图 4-70 所示。

序号	时间	用户ID	交易类型	收入	支出	备注
1	2018-04-25 08:51:57	coinbase_01234567	提现手续费	0.000000	0.000000	提现手续费
2	2018-04-25 08:52:57	coinbase_01234567	提现手续费	0.000000	0.000000	提现手续费
3	2018-04-25 08:53:57	coinbase_01234567	提现手续费	0.000000	0.000000	提现手续费
4	2018-04-25 08:54:57	coinbase_01234567	提现手续费	0.000000	0.000000	提现手续费
5	2018-04-25 08:55:57	coinbase_01234567	提现手续费	0.000000	0.000000	提现手续费
6	2018-04-25 08:56:57	coinbase_01234567	提现手续费	0.000000	0.000000	提现手续费

图 4-70





平台虚拟币交易记录表结构如下:

```
REATE TABLE `PlatFromTokenTransInfo` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '自增 ID',
  `coinId` int(10) DEFAULT NULL COMMENT '币种 ID, 参考 CoinInfo.id',
  `type` varchar(10) DEFAULT NULL COMMENT '交易类型',
  `time` timestamp NULL DEFAULT NULL COMMENT '交易时间',
  `income` decimal(20,8) DEFAULT '0.00000000' COMMENT '收入',
  `payout` decimal(20,8) DEFAULT '0.00000000' COMMENT '支出',
  `balance` decimal(20,8) DEFAULT '0.00000000' COMMENT '余额',
  `referenceId` int(10) unsigned DEFAULT NULL COMMENT '关联 ID',
  `remark` varchar(50) DEFAULT '' COMMENT '备注',
  `userAccountId` int(11) DEFAULT NULL COMMENT '用户 ID, 参考 UserAccount.id',
  PRIMARY KEY (`id`),
  KEY `userAccountId` (`userAccountId`) USING BTREE,
  KEY `coinId` (`coinId`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=22 DEFAULT CHARSET=utf8 COMMENT='平台虚拟币交易记录(PlatFromTokenTransInfo)';
```

### (3) 平台虚拟币钱包

平台虚拟币钱包是指平台每种虚拟币的提币钱包,如图 4-71 所示。当用户发起转出资产请求,并且在虚拟币转出申请通过后,系统会从平台虚拟币钱包向用户提交的提币钱包中转入相应数量的虚拟币。

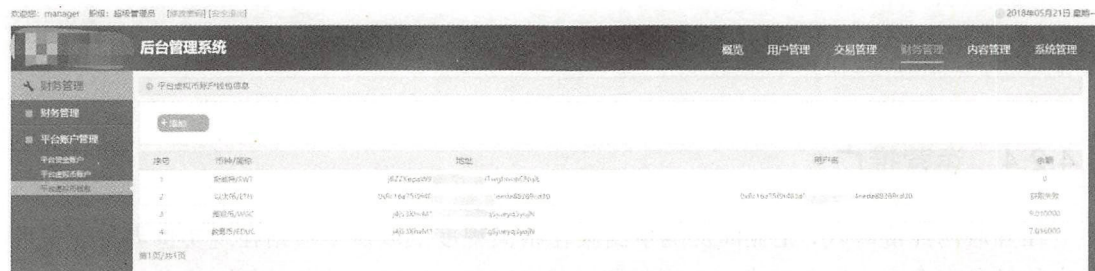


图 4-71

“添加平台虚拟币钱包”页面如图 4-72 所示。

创建一个虚拟币的提币钱包地址,若不是以太坊系的币种,则只需要填写地址和私钥即可。如果是以太坊系的币种,则需要查询链上的钱包地址余额,所以图 4-72 中的输入框均需要填写。



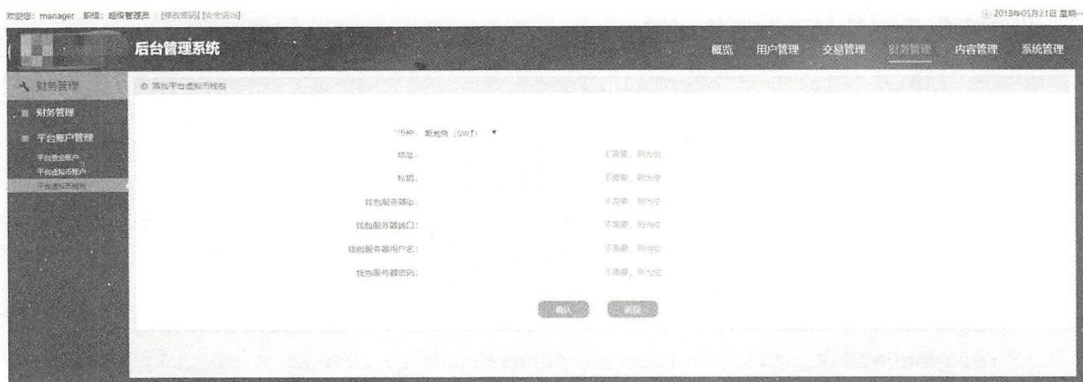


图 4-72

平台虚拟币钱包表结构如下：

```
CREATE TABLE `PlatTokenWallet` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '自增 ID',
  `coinId` int(11) DEFAULT '0' COMMENT '币种 ID, 参考 CoinInfo.id',
  `walletAdress` varchar(50) DEFAULT NULL COMMENT '平台钱包地址',
  `scretKey` text COMMENT '私钥',
  `serverIP` varchar(30) DEFAULT NULL COMMENT '钱包服务器 IP 地址',
  `serverPort` varchar(10) DEFAULT NULL COMMENT '钱包服务器端口',
  `serverUserName` varchar(100) DEFAULT NULL COMMENT '钱包服务器用户名',
  `serverPassword` text COMMENT '钱包服务器密码',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8 COMMENT='平台虚拟币钱包(PlatTokenWallet)';
```

## 4.2.4 运营推广

在互联网经济时代，企业间的竞争结果在很大程度上取决于网站在现实世界与虚拟世界中的触角有多少，触角越多、力度越强，就可吸引更多的关注与访问。为此，企业在建立网站后即应着手利用各种手段推广自己的网站。网站推广是一个循序渐进的过程，需要经过市场培育，不断发展和完善。

### 1. 派发“糖果”

很多刚进币圈的朋友不知道派发“糖果”是什么意思，其实派发“糖果”就是指针针对



持有某些特定币种的用户，根据一定比例赠送其他币种。其他币种可以卖掉，“糖果”就是钱，派发红包。在项目发展之初，为了获得用户、人气等会免费发放一些福利，类似于手机 App 刚上线时注册就送红包，就像在腾讯发展早期使用 QQ 就送 Q 币一样。在区块链的用语中，统称赠送的东西为“糖果”。

其实在我们熟知的一些数字货币中，很多都是以免费或极低的费用发放给最初的用户。比如比特币刚出来时挖矿非常容易，价格只有几美分，以至于有些人挖出来之后存在硬盘里就忘记这件事了，后来把硬盘丢了；有的人用 10000 比特币换了两个比萨，这些比特币按照现在的价格来算相当于 1 亿美元，这两个比萨被称为史上最贵比萨。还有前段时间赚足眼球的瑞波币，由于数量比较多，刚开始时也是大量往外送。这样的行为就称为“派发糖果”。

除此之外，还有一种推广手段就是邀请奖励，即邀请一个用户可以得到什么奖励。这也是很多网站在使用的推广手段。比如之前浏览过的一个网站，它的推广方案是这么写的：

### 去中心化运营 开启新时代

各位亲爱的用户：

为了弘扬数字经济去中心、分布式的特点，本平台确定采取去中心化的运营方式，希望社会各界数字经济爱好者共同参与、共同发展。具体方式如下：

(1) 成功引荐新币种/token 上线本平台的，将获得该币种/token 交易手续费 60% 的分成。

(2) 成功邀请新用户注册的，将获得该用户交易手续费 30% 的分成。

(3) 本平台负责运营维护、技术支持、安全防护、更新升级等运维工作，获得平台交易手续费 10% 的分成。

(4) 本平台不收取任何上币费用；邀请凭证为用户注册时填写的邀请码（邀请链接）。

(5) 手续费分成生效日期：2018 年 5 月 11 日，第一批结算将于 6 月 10 日兑现，之后每周结算一次，直接转入用户在平台的账户中。

XXX 致力于提供一个开放、真实、安全的共享平台，降低数字资产交易成本，打破





不公平的利益分配方式，实现全民共享、共生、共建的去中心化运营方式，推动数字资产的流通和应用。

www.\*\*\*.com \*\*\*网

2018年5月10日

## 2. 投票

有时平台需要统计用户的需求，或者征求用户对某个币种上线的意见，这时我们就需要一个投票系统。投票系统的具体代码这里不再赘述，网络上有很多搭建投票系统的教程，有兴趣的读者可以参考，然后根据自身平台的需求进行设计开发。

### 4.2.5 系统监控及预警

一个网站的正常运行，离不开安全保护，以及对系统异常的监控与预警。一般可以采取如下措施。

#### (1) 数据库定期备份

在互联网高度发展的今天，数据不论是对企业还是用户都是至关重要的，当数据发生问题时会给企业造成不可估量的损失。通常对数据的威胁比较难于防范，一旦威胁变为现实，则不仅会丢失或毁坏数据，也会毁坏访问数据的系统。为了最大限度地减少企业损失，平台应每日定时对数据进行全量备份，为数据还原及恢复做好充分保障。

#### (2) 日志留存

平台不仅在接口、数据交互等方面进行了错误日志的抓取，而且也保存了服务器日常运行中的必要信息，可以通过日志快速定位问题，也可以在日志中进行关键数据的抓取，支持大数据分析，并且平台对日志的留存时间在30天左右。

#### (3) 资产稽核系统

资产稽核系统不仅提升了平台用户资产的安全性，而且是对企业权益的最大保障，针对非法用户利用网络漏洞对资产余额、交易等进行篡改、删除等操作进行及时的定位及控制，减少用户及企业损失，保护利益共同体健康成长。



#### (4) 用户黑名单

当发现用户资产受到威胁,或者给企业带来损失时,平台可自动或手动对用户进行锁定,将其添加至黑名单中,被锁定的用户将无法继续进行交易,保障了资产的安全性。

#### 1. 网站流量统计

网站流量统计可以准确地分析访客的来源,便于网站管理者根据访客的需求增加或者修改网站的相关内容,更好地提升网站转换率,提高网站流量。网站流量统计的作用如下:

- ◎ 精确地统计访客的具体来源地区和 IP 地址。
- ◎ 精确地统计目前网站在线多少人,以及他们具体访问了哪些页面。
- ◎ 精确地统计访客是通过哪些页面搜索关键词访问网站的,以及访客浏览的是哪些页面。
- ◎ 精确地统计访客的浏览器版本。
- ◎ 精确地统计网站的浏览量。
- ◎ 分时统计、分日统计、分月统计,实时统计用户访问最多的页面。
- ◎ 精确地统计访客的操作系统是什么、分辨率是多少。

网站流量统计在营销中的意义如下:

- ◎ 及时掌握网站推广的效果,减少盲目性。
- ◎ 分析各种网络营销手段的效果,为制定和修正网络营销策略提供依据。
- ◎ 通过网站访问数据分析进行网络营销诊断,包括对各项网站推广活动的效果分析、网站优化状况诊断等。
- ◎ 了解用户访问网站的行为,为更好地满足用户需求提供支持。
- ◎ 可以作为网络营销效果评价的参考指标。
- ◎ 帮助了解网站的访问情况,提前应对系统负荷问题。
- ◎ 根据监测到的访问客户端的信息来优化网站设计和功能。

由此可见,对网站流量进行统计是十分必要的。我们需要在数据库中建立相应的表来



存储流量统计数据，在处理类中进行统计。一般就是利用 Servlet 获取 Request 对象中的信息，获取访问用户的 IP 地址、所使用的浏览器、系统、访问来源等，将其存储到数据库中。在前台页面中可以设计统计功能用来统计，也可以直接利用一些网站流量统计工具，如百度统计、GA、CNZZ 等。

## 2. 异常监控

对异常进行监控及预警，对于保障平台的资金安全是十分重要的。一般会设计一个对账系统，使用定时器不间断地扫描平台及用户的资金变化情况。如果出现异常，则发送邮件到指定账户，同时对出现问题的账户进行锁定，将用户及平台的损失最小化。

异常监控主要涉及以下几个地方。

### (1) 用户 UST 记录

UST 余额（可用的 UST+冻结的 UST）= C2C 成功买入的 UST-交易购买支出的 UST+交易卖出虚拟币收入的 UST-C2C 成功卖出的 UST（申请卖出金额）。

#### ①用户 UST 余额。

```
SELECT sum(freezeAmount) '冻结金额',sum(useableAmount) '可用金额' FROM
UserCNYInfo WHERE 1 = 1 and id= ? #用户 ID
```

#### ②用户 UST 余额（按时间结余）。

```
SELECT sum( PlatFormTransInfo.income) '收入', sum(PlatFormTransInfo.
payOut) '支出',PlatFormTransInfo.balance '结余',
UserAccount.userName AS USERNAME,UserInfo. xm,CoinInfo.nameCn nameCn,
PlatFormTransInfo.income
FROM ZCH_S70.PlatFormTransInfo LEFT JOIN ZCH_S60.UserAccount ON
PlatFormTransInfo.userId=UserAccount.id
INNER JOIN ZCH_S60.UserInfo ON UserAccount.id=UserInfo.userAccountId
LEFT JOIN ZCH_S60.CoinInfo ON PlatFormTransInfo.coinId=CoinInfo.id
WHERE 1=1
AND PlatFormTransInfo.coinId = 1 #币种 ID
AND PlatFormTransInfo.userId = 3 #用户 ID
AND PlatFormTransInfo.createTime between '2018-03-20 15:00:55' and
'2018-04-13 22:00:55' #时间范围
group by PlatFormTransInfo.createTime Order by PlatFormTransInfo.F04 desc
limit 0,1 #按时间排序获取最新的信息
```



## ③C2C 成功买入。

```
SELECT  sum(actulToAccount) buy  from C2CTransInfo
WHERE 1 = 1
AND status = 3 #交易完成
AND  userId = ? #用户 ID
AND type = 1 #买入类型
AND createTime between '2018-03-20 15:00:55' and '2018-04-13 15:00:55' #时间范围
```

## ④C2C 成功卖出。

```
SELECT  sum(actulToAccount) sell  from C2CTransInfo
WHERE 1 =1
AND type = 3 #交易完成
AND  userId = ? #用户 ID
AND type = 1 #买入类型
AND createTime between '2018-03-20 15:00:55' and '2018-04-13 15:00:55' #时间范围
```

## ⑤交易收入 UST (type= MC)。

查询在 UST 分区下所有卖出收入的 UST 金额与支出的 UST 手续费。

```
SELECT sum(UserTransInfo.) total,sum(UserTransInfo.F09) sellFees,
sum(UserTransInfo.fee) - sum(UserTransInfo.income) incom
FROM UserTransInfo
LEFT JOIN marketInfo ON UserTransInfo.userAccountId=marketInfo.id
LEFT JOIN UserAccount  ON UserTransInfo.coinId=UserAccount.id
WHERE 1=1
AND UserTransInfo.type = 'MC' #交易类型
AND marketInfo.coinId = 1 #买方币种 ID (UST 分区)
AND UserTransInfo.createTime between '2018-03-20 15:00:55' and '2018-04-13 15:00:55' #时间范围
```

## ⑥交易支出 UST (type= MR)。

查询在 UST 分区下，用户买入虚拟币消耗的 UST 总金额。因为买入的手续费扣除的不是 UST，所以手续费不计算在内。

```
SELECT sum(UserTransInfo.fee) pay
FROM UserTransInfo
LEFT JOIN CoinInfo buy ON buy.id=marketInfo.buy_coin
LEFT JOIN UserAccount buy_id ON buy_id.id=UserTransInfo.coinId
```

```
WHERE 1=1
AND UserTransInfo.type = 'MR' #交易类型
AND marketInfo.coinId = 1 #买方币种 ID (UST 分区)
AND UserTransInfo.createTime between '2018-03-20 15:00:55' and '2018-04-13
15:00:55' #时间范围
```

## (2) 用户虚拟币

例如计算 MAE 余额：MAE 余额=转入数量-转出数量-作为买方币种时买入的金额+作为买方币种时卖出的数量+作为卖方币种时买入的数量-作为卖方币种时卖出的金额-作为买方的手续费-作为卖方的手续费+赠送的虚拟币。

### ①MAE 余额。

全部用户余额（虚拟币）：

```
SELECT sum( freezeAmount) '冻结数量', sum(useableAmount) '可用' FROM
UserCoinInfo WHERE 1 = 1
AND userId=3 #用户 ID
AND coinId= 2 #币种 ID
```

按时间结余：

```
SELECT sum(PlatFormTransInfo.income) '收入', sum(PlatFormTransInfo.payOut)
'支出', PlatFormTransInfo.balance '结余', UserAccount.userName AS USERNAME,
UserInfo.name name, CoinInfo.nameCn nameCn, PlatFormTransInfo.TransTime
FROM ZCH_S70.PlatFormTransInfo LEFT JOIN ZCH_S60.UserAccount ON
PlatFormTransInfo.userId=UserAccount.id
INNER JOIN ZCH_S60.UserInfo ON UserAccount.userId=UserInfo.id
LEFT JOIN ZCH_S60.CoinInfo ON PlatFormTransInfo.TransType=CoinInfo.id
WHERE 1=1
AND PlatFormTransInfo.coinId = 2 #币种 ID
AND PlatFormTransInfo.userId = ? #用户 ID
AND PlatFormTransInfo.createTime between '2018-03-20 15:00:55' and
'2018-04-13 22:00:55' #时间范围
group by PlatFormTransInfo.createTime Order by PlatFormTransInfo. createTime
desc limit 0,1
```

### ②转入数量。

```
SELECT sum(UserCoinRollIn.amount) '转入数量', sum(USerCoinRollIn.actulToAccount) '
实际到账' FROM USerCoinRollIn LEFT JOIN CoinInfo ON USerCoinRollIn.coinId=
CoinInfo.id
```



```

WHERE 1=1
AND UserCoinRollIn.coinId=2 #币种 ID
AND UserCoinRollIn.UserId=3 #用户 ID
AND UserCoinRollIn.status = 's' #转入成功
AND UserCoinRollIn.createTime between '2018-03-20 15:00:55' and '2018-04-13
15:00:55'
ORDER BY UserCoinRollIn.TransTime DESC

```

### ③转出数量。

```

SELECT sum(UserCoinRollOut.actultoAccount) '实际到账',sum(UserCoinRollOut.
serviceCharge) '手续费',CoinInfo.nameEn'英文名',CoinInfo.nameCn '中文名',
UserCoinRollOut.status '状态'
FROM UserCoinRollOut LEFT JOIN CoinInfo ON UserCoinRollOut.coinId=CoinInfo.id
LEFT JOIN UserWalletInfo ON UserCoinRollOut.walletAdress=UserWalletInfo .id
WHERE 1=1
AND UserCoinRollOut.coinId=2 #币种 ID
AND UserCoinRollOut.userId=3 #用户 ID
AND UserCoinRollOut.rollOutSuccess = 's' AND UserCoinRollOut.status = 'ZCCG'
#转出成功
AND UserCoinRollOut.time between '2018-03-20 15:00:55' and '2018-04-13
15:00:55' #时间范围
ORDER BY UserCoinRollOut.time DESC

```

实际到账数量即为转出数量。

### ④作为买方币种时买入的金额（手续费扣其他币种）。

```

SELECT sum(UserTransInfo.fee) pay #支出
FROM UserTransInfo
LEFT JOIN marketInfo ON UserTransInfo.marketId=marketInfo.id
LEFT JOIN UserAccount ON UserTransInfo.userAccountId=UserAccount.id
LEFT JOIN CoinInfo buy ON buy.marketId=marketInfo.id
LEFT JOIN UserAccount buy_id ON buy_id.id=UserTransInfo. userId
WHERE 1=1
AND UserTransInfo.type= 'MR' #交易类型
AND marketInfo.coinId = 2 #资产通作为买方币种
AND UserTransInfo.time between '2018-03-20 15:00:55' and '2018-04-13
15:00:55' #时间范围

```

### ⑤作为买方币种时卖出的数量（扣除作为买方的手续费）。

```

SELECT sum(UserTransInfo.totalIncome) '总金额',sum(UserTransInfo.serviceChage)

```



## 区块链：交易系统开发指南

```

'手续费',sum(UserTransInfo.totalIncome)-sum(UserTransInfo.serviceChage) '实际
收入'
FROM UserTransInfo
LEFT JOIN marketInfo ON UserTransInfo.merketId=marketInfo.id
LEFT JOIN UserAccount ON UserTransInfo.userId=UserAccount.id
LEFT JOIN CoinInfo sell ON sell.id=marketInfo.sell_coin
LEFT JOIN UserAccount sell_id ON sell_id.id=UserTransInfo.sell_coin
WHERE 1=1
AND UserTransInfo.type= 'MC' #交易类型
AND marketInfo.coinId = 2 #资产通作为买方币种
AND UserTransInfo.time between '2018-03-20 15:00:55' and '2018-04-13
15:00:55' #时间范围

```

⑥作为卖方币种时买入的数量（扣除作为卖方的手续费）。

```

SELECT sum(UserTransInfo.totalAmount) '总金额',sum(UserTransInfo.serviceCharge)
'手续费',sum(UserTransInfo.totalAmount)-sum(UserTransInfo.serviceCharge) '实际
收入'
FROM UserTransInfo
LEFT JOIN marketInfo ON UserTransInfo.marketId=marketInfo.id
LEFT JOIN UserAccount ON UserTransInfo.userId=UserAccount.id
LEFT JOIN CoinInfo buy ON buy.id=marketInfo.buy_coinId
LEFT JOIN UserAccount buy_id ON buy_id.userId=UserTransInfo.userId
WHERE 1=1
AND UserTransInfo.type= 'MR' #交易类型
AND marketInfo.coinId = 2 #资产通作为卖方币种
AND UserTransInfo.time between '2018-03-20 15:00:55' and '2018-04-13
15:00:55' #时间范围

```

⑦作为卖方币种时卖出的金额（手续费扣其他币种）。

```

SELECT sum(UserTransInfo.payOut) '支出'
FROM UserTransInfo
LEFT JOIN marketInfo ON UserTransInfo.marketId=marketInfo.id
LEFT JOIN UserAccount ON UserTransInfo.userId=UserAccount.id
LEFT JOIN CoinInfo sell ON sell.id=marketInfo.sell_coinId
LEFT JOIN UserAccount sell_id ON sell_id.id=UserTransInfo.sell_userId
WHERE 1=1
AND UserTransInfo.type= 'MC' #交易类型
AND marketInfo.coinId = 2 #资产通作为卖方币种
AND UserTransInfo.time between '2018-03-20 15:00:55' and '2018-04-13
15:00:55' #时间范围

```

## ⑧赠送的虚拟币。

```

SELECT  sum(PlatTransInfo.presentCoin) '虚拟币赠送', UserAccount.username
AS 'username',UserInfo.name '用户名'
FROM ZCH_S70.PlatTransInfo LEFT JOIN
ZCH_S60.UserAccount ON PlatTransInfo.userId=UserAccount.id
INNER JOIN ZCH_S60.UserInfo ON UserAccount.id=UserInfo.userId
WHERE 1=1
AND PlatTransInfo.userId = ? #用户 ID
AND PlatTransInfo.status = 'ZS'
AND PlatTransInfo.time between '2018-03-20 15:00:55' and '2018-04-13
15:00:55' #时间范围

```

## (3) 虚拟币总账单

赠送的虚拟币+转入数量-转出数量=平台手续费收入+全部用户余额（可用金额+冻结数量）。

## ①总赠送的虚拟币。

```

SELECT  sum(PlatTransInfo.presentCoin) '虚拟币赠送', UserAccount.username
AS 'username',UserInfo.name '用户名'
FROM ZCH_S70.PlatTransInfo LEFT JOIN
ZCH_S60.UserAccount ON PlatTransInfo.userId=UserAccount.id
INNER JOIN ZCH_S60.UserInfo ON UserAccount.id=UserInfo.id
WHERE 1=1
AND PlatTransInfo.type= 'JYSXF'
AND PlatTransInfo.time between '2018-03-20 15:00:55' and '2018-04-13
15:00:55' #时间范围

```

## ②总转入数量。

```

SELECT  sum(USerCoinRollIn.rollInAmount) '转入数量', sum(USerCoinRollIn.
actulToAccount) '实际到账',USerCoinRollIn.F02 '用户 ID'
FROM USerCoinRollIn
LEFT JOIN CoinInfo ON USerCoinRollIn.coinId=CoinInfo.id
WHERE 1=1
AND USerCoinRollIn.coinId = 2 #币种 ID
AND USerCoinRollIn.status = 's' #转入成功
AND USerCoinRollIn.time between '2018-03-20 15:00:55' and '2018-04-13
15:00:55'
ORDER BY USerCoinRollIn.time DESC

```





### ③总转出数量。

```
SELECT sum(UserCoinRollOut.actulToAccount) '实际到账', sum(UserCoinRollOut.
serviceCharge) '手续费', CoinInfo.nameEn '英文名', CoinInfo.nameCn '中文名',
UserCoinRollOut.status '状态'
FROM UserCoinRollOut LEFT JOIN CoinInfo ON UserCoinRollOut.coinId=
CoinInfo.id
LEFT JOIN UserWalletInfo ON UserCoinRollOut.walletid=UserWalletInfo.id
WHERE 1=1
AND UserCoinRollOut.coin=2 #币种 ID
AND UserCoinRollOut.is_success= 's' AND UserCoinRollOut.status= 'ZCCG' #
转出成功
AND UserCoinRollOut.time between '2018-03-20 15:00:55' and '2018-04-13
15:00:55' #时间范围
ORDER BY UserCoinRollOut.time DESC
```

### ④平台手续费收入，即转出手续费+交易手续费。

#### 转出手续费：

```
SELECT sum(UserCoinRollOut.actulToAccount) '实际到账', sum(UserCoinRollOut.
serviceCharge) '手续费',
CoinInfo.nameEn '英文名', CoinInfo.nameCn '中文名', UserCoinRollOut.status '
状态'
FROM UserCoinRollOut LEFT JOIN CoinInfo ON UserCoinRollOut.coinId=
CoinInfo.id
LEFT JOIN UserWalletInfo ON UserCoinRollOut.walletId=UserWalletInfo.id
WHERE 1=1
AND UserCoinRollOut.coinId =2 #币种 ID
AND UserCoinRollOut.is_success = 's' AND UserCoinRollOut.status = 'ZCCG' #
转出成功
AND UserCoinRollOut.time between '2018-03-20 15:00:55' and '2018-04-13
15:00:55' #时间范围
ORDER BY UserCoinRollOut.time DESC
```

#### 交易手续费（作为卖方币种买入）：

```
SELECT sum(UserTransInfo.serviceCharge) '手续费'
FROM UserTransInfo
LEFT JOIN marketInfo ON UserTransInfo.marketId=marketInfo.id
LEFT JOIN UserAccount ON UserTransInfo.userId=UserAccount.id
LEFT JOIN CoinInfo buy ON buy.id=marketInfo.buy_coinId
LEFT JOIN UserAccount buy_id ON buy_id.id=UserTransInfo.buy_userId
```





```
WHERE 1=1
AND UserTransInfo.type = 'MR' #交易类型
AND marketInfo.coinId = 2 #资产通作为卖方币种
AND UserTransInfo.time between '2018-03-20 15:00:55' and '2018-04-13
15:00:55' #时间范围
```

交易手续费（作为买方币种卖出）：

```
SELECT sum(UserTransInfo.serviceCharge) '手续费'
FROM UserTransInfo
LEFT JOIN marketInfo ON UserTransInfo.marketId=marketInfo.id
LEFT JOIN UserAccount ON UserTransInfo.userId=UserAccount.id
LEFT JOIN CoinInfo sell ON sell.id=marketInfo.sell_coinId
LEFT JOIN UserAccount sell_id ON sell_id.id=UserTransInfo.sell_userId
WHERE 1=1
AND UserTransInfo.type= 'MC' #交易类型
AND marketInfo.coinId= 2 #资产通作为买方币种
AND UserTransInfo.time between '2018-03-20 15:00:55' and '2018-04-13
15:00:55' #时间范围
```

⑤全部用户余额（虚拟币）。

```
SELECT sum( freezeAmount) '冻结数量', sum(useableAmount) '可用' FROM
UserCoinInfo WHERE 1 = 1 AND coinId= 2 #币种 ID
```

按时间结余：

```
SELECT sum( PlatFormTransInfo.income) '收入', sum(PlatFormTransInfo.payOut)
'支出',PlatFormTransInfo.balance '结余',
UserAccount.userNmae AS USERNAME,UserInfo.name name,CoinInfo.nameCn nameCn,
PlatFormTransInfo.income FROM ZCH_S70.PlatFormTransInfo LEFT JOIN ZCH_S60.
UserAccount ON PlatFormTransInfo.userId=UserAccount.id
INNER JOIN ZCH_S60.UserInfo ON UserAccount.id=UserInfo.id
LEFT JOIN ZCH_S60.CoinInfo ON PlatFormTransInfo.coinId=CoinInfo.id
WHERE 1=1
AND PlatFormTransInfo.coinId= 2 #币种 ID
AND PlatFormTransInfo.time between '2018-03-20 15:00:55' and '2018-04-13
22:00:55' #时间范围
group by PlatFormTransInfo.createTime Order by PlatFormTransInfo.time desc
limit 0,1
```



#### (4) UST 总账单

C2C 买入-C2C 卖出=平台手续费收入+全部用户余额（可用金额+冻结数量）。

##### ①C2C 买入。

```
SELECT sum(actulToAccount*exchangeRate) from C2CTransInfo
WHERE 1 = 1
AND status = 3 #交易完成
AND type = 1 #买入类型
AND time between '2018-03-20 15:00:55' and '2018-04-13 15:00:55' #时间范围
```

##### ②C2C 卖出。

```
SELECT sum(actulToAccount*exchangeRate) from C2CTransInfo
WHERE 1 = 1
AND status = 3 #交易完成
AND type = 2 #卖出类型
AND time between '2018-03-20 15:00:55' and '2018-04-13 15:00:55' #时间范围
```

##### ③平台手续费收入。

只有 UST 分区，卖出时手续费：

```
SELECT sum(UserTransInfo.serviceCharge) '手续费'
FROM UserTransInfo
LEFT JOIN marketInfo ON UserTransInfo.marketId=marketInfo.id
LEFT JOIN UserAccount ON UserTransInfo.userId=UserAccount.id
LEFT JOIN CoinInfo sell ON sell.id=marketInfo.sell_coinId
LEFT JOIN UserAccount sell_id ON sell_id.F01=UserTransInfo.sell_userId
WHERE 1=1
AND UserTransInfo.type = 'MC' #交易类型
AND marketInfo.coinid = 1 #UST 作为买方币种
AND UserTransInfo.time between '2018-03-20 15:00:55' and '2018-04-13
15:00:55' #时间范围
```

##### ④全部用户余额（UST）。

```
SELECT sum(freezeAmount) '冻结金额',sum(useableAmount) '可用金额' FROM
UserUSTInfo
WHERE 1 = 1
```

按时间结余：

```
SELECT sum(PlatFormTransInfo.income) '收入', sum(PlatFormTransInfo.payOut)
```



```
'支出',PlatFormTransInfo.balance '结余',
    UserAccount.username AS USERNAME,UserInfo.name name,CoinInfo.nameEn nameRn,
PlatFormTransInfo.time
    FROM ZCH_S70.PlatFormTransInfo LEFT JOIN ZCH_S60.UserAccount ON
PlatFormTransInfo.userId=UserAccount.id
    INNER JOIN ZCH_S60.UserInfo ON UserAccount.id=UserInfo.id
    LEFT JOIN ZCH_S60.CoinInfo ON PlatFormTransInfo.coinId=CoinInfo.id
WHERE 1=1
AND PlatFormTransInfo.coinId = 1 #币种 ID
AND PlatFormTransInfo.time between '2018-03-20 15:00:55' and '2018-04-13
22:00:55' #时间范围
group by PlatFormTransInfo.time Order by PlatFormTransInfo.time desc limit 0,1
```

### 3. 稽核报表

系统在每天结束时都会将平台的资金变化情况，以及平台用户的总资金变化情况，以短信的形式发送给负责人。使用定时器及邮件发送系统将稽核的内容汇总发出即可。

## 4.3 多语言

### 4.3.1 多语言的目的

通俗地说，打开一个国外网站却以为是国内公司创建的，那么这个网站的多语言处理就非常成功。所以，多语言的目的就是让各个地区的人都能感受到这个网站就像本地网站一样，亲切、易用。

多语言处理主要分为两个部分。

#### (1) 多语言内容的改变

内容主要包括固定文案和动态数据。

#### (2) 多语言样式的改变

- ◎ 根据语系和习惯来定义 class，而不是语言。比如英语和德语都是字母，它们的样式个性化就有共通性，而不需要定义多个 class 重复编写规则，避免产生冗余代码。





- ◎ 样式是用来适配内容而不是约束内容的。将中文内容翻译成英文，长度势必会增长，从而产生错位。

### 4.3.2 多语言网站实现方案

#### 1. 实现多语言网站的两种方案

静态方案：就是为每种语言分别准备一套页面文件，要么通过文件后缀来区分不同的语言，要么通过子目录来区分不同的语言。

例如，对于首页文件 `index_en.htm` 提供英语界面，`index_gb.htm` 提供简体中文界面，`index_big.htm` 提供繁体中文界面，或者 `en/index.htm` 提供英语界面，`gb/index.htm` 提供简体中文界面，`big/index.htm` 提供繁体中文界面。一旦用户选择了语言，就会自动跳转到相应的页面。从维护的角度来看，通过子目录比通过文件后缀来区分不同的语言要简单明了。

动态方案：在站点内所有页面文件都是动态的（PHP、ASP 等），而不是静态的，在需要输出语言文字的地方统一采用语言变量来表示，语言变量可以根据用户选择的不同语言赋予不同的值，从而能够实现在不同的语言环境下输出不同的文字。

例如语言变量 `ln_name`，当用户选择的语言是英语时赋值为“Name”，当用户选择的语言是简体中文时赋值为“姓名”，这样就可以适应不同语言的输出。

#### 2. 两种方案的优缺点

采用静态方式的优点是页面直接输出到客户端，不需要在服务器上运行，占用服务器的资源比较少，系统能够支持的并发连接数较多；缺点是要为每种语言都设计一套页面文件，很多内容即使和语言无关也要分不同语言来存储，因此占用的存储空间较多。

采用动态方式的优点是动态页面文件只有一套，不同语言的文字使用语言变量来存储，和语言无关的内容只存储一份，占用的存储空间较少，并且扩展新语言比较容易；缺点是需要服务器上运行，然后把结果输出到客户端，占用服务器的资源比较多，系统能够支持的并发连接数较少。



### 3. 动态数据存储涉及的一些技术问题

现在网站上的动态应用日益增多,有相当多的网站还在使用文件或者数据库来存储应用信息,因此当文件或者数据库中存储的内容与语言相关时需要特别注意。对于存储在数据库中的信息,可以通过以下几种方式来支持多语言。

- ◎ 在数据库级别支持多语言:为每种语言建立独立的数据库,不同语言的用户操作不同的数据库。
- ◎ 在表级别支持多语言:在同一个数据库中为每种语言建立独立的表,不同语言的用户操作不同的表。
- ◎ 在字段级别支持多语言:在同一个表中为每种语言建立独立的字段,不同语言的用户操作不同的字段。

由于数据库中有大量的信息(如标志、编码、数字等)是用于内部处理的,与语言无关,因此在数据库级别支持多语言会导致空间的极大浪费,在字段级别支持多语言存在的最大问题是一旦需要支持新的语言,就需要修改表结构,维护起来非常麻烦,可扩展性不好。

相比之下,在表级别支持多语言比较好,因为并不是所有的表都需要支持多语言,对于与语言无关的表,不同语言的用户可以共用;对于与语言相关的表,可以根据支持语言的种类来建立,不同语言的用户存取访问不同的表。这样做就会使得维护简单,节省了存储空间,扩展也比较方便,只需把支持多语言的表多建立一套即可。

另外,需要注意的是,有些表中的某些字段是不同语言的表共享的(例如库存量),由于各种语言的表之间的相对独立性,使得数据共享有些困难。

解决方法有两个。

- ◎ 不同语言的表的共享字段同步:也就是说,只要修改了其中一个表的共享字段,其他语言的表中的该字段也做相应的改变。实际上,当不同语言的用户同时访问时处理还是比较麻烦的,并且在扩展新语言时的修改工作量比较大。
- ◎ 增加一个新的表:把所有语言的表的共享字段(例如货物编号、产地编码等)全部放在这个表中,支持多语言的表只存放与各种语言相关的字段。不同语言的用户在使用数据库时,需要操作两个表。



#### 4. 在前端开发中进行多语言切换，可使用JavaScript动态替换内容

(1) 用户点击切换语言后，把所选择的语言版本保存在 Cookie 中

```
// 写入 Cookie 函数
function setCookie(name,value){
    var Days = 30;
    var exp = new Date();
    exp.setTime(exp.getTime() + Days*24*60*60*1000);
    document.cookie = name + "=" + escape (value) + ";expires=" +
exp.toGMTString();
}
// 获取 Cookie
function getCookie(name){
    var arr,reg=new RegExp("(^| )" +name+"=([^;]*) (;|$)");
    if(arr=document.cookie.match(reg))
        return unescape(arr[2]);
    else
        return null;
}
// setCookie('lan','hk');    繁体中文
// setCookie('lan','cn');    简体中文
// setCookie('lan','en');    英文
```

(2) 在包含静态文本的标签中添加一个属性：set-lan="html:name"

属性值中的 html 代表内容的位置，name 代表要替换的文字标识。例如：

```
<span set-lan="html:name">名字</span>
<input type="text" value="名字" set-lan="val:name" />
```

这需要前端工程师在开发过程中将其添加进去，并且在要切换语言文字的外面都必须有一个标签包裹，否则无法进行切换。例如：

```
<span set-lan="html:name"><i class="icon"></i>名字</span>
```

这样的代码是无法把“名字”替换成“Name”的，会把“名字”前面的 i 标签也一起替换掉。如果不想把 i 标签也替换掉，就要在“名字”外面添加一个标签，改成：

```
<span><i class="icon"></i><lan set-lan="html:name">名字</lan></span>
```

(3) 定义 3 个语言标识+内容的 JSON 字符串

```
var cn = {
```





```

        "name" : "姓名",
        "tel" : "电话",
        "email" : "邮箱",
    };
    var hk = {
        "name" : "姓名",
        "tel" : "电话",
        "email" : "邮箱",
    };
    var en = {
        "name" : "Name",
        "tel" : "Tel",
        "email" : "Email",
    };

```

#### (4) 遍历带 set-*lan* 属性的标签, 进行文本替换

```

$('[set-lan]').each(function(){
    var me = $(this);
    var a = me.attr('set-lan').split(':');
    var p = a[0]; // 文字放置位置
    var m = a[1]; // 文字标识
    // 用户选择语言后保存在 Cookie 中, 这里读取 Cookie 中的语言版本
    var lan = getCookie('lan');
    // 选择语言文字
    switch(lan){
        case 'cn':
            var t = cn[m]; // 这里 cn[m] 中的 cn 是上面定义的 JSON 字符串的变量名
                           // m 是 JSON 中的键, 用此方式读取到 JSON 中的值
            break;
        case 'en':
            var t = en[m];
            break;
        default:
            var t = hk[m];
    }
    // 如果所选语言的 JSON 中没有此内容, 就选择其他语言显示
    if(t==undefined) t = cn[m];
    if(t==undefined) t = en[m];
    if(t==undefined) t = hk[m];
    if(t==undefined) return true; // 如果还是没有, 就跳出

```



```
// 文字放置位置有 html、val 等，可以自己添加
switch(p) {
    case 'html':
        me.html(t);
        break;
    case 'val':
    case 'value':
        me.val(t);
        break;
    default:
        me.html(t);
}
});
```

以上是 HTML 文件中的文字替换，但是写在 JavaScript 文件中的文字怎么办？

可以定义一个函数来读取：

```
function get_lan(m) {
    // 获取文字
    var lan = getCookie('lan');    // 语言版本
    // 选择语言文字
    switch(lan) {
        case 'cn':
            var t = cn[m];
            break;
        case 'hk':
            var t = hk[m];
            break;
        default:
            var t = en[m];
    }

    // 如果所选语言的 JSON 中没有此内容，就选择其他语言显示
    if(t==undefined) t = cn[m];
    if(t==undefined) t = en[m];
    if(t==undefined) t = hk[m];
    if(t==undefined) t = m; // 如果还是没有，就返回其标识
    return t;
}
```



写在 JavaScript 文件中的文字只要用此函数来读取就可以了。比如将

```
alert('姓名');
```

改成

```
alert(get_lan('name'));
```

## 4.4 软件安全测试

### 4.4.1 安全测试基本概念

软件安全测试包括程序、网络、数据库安全测试。根据系统安全指标的不同，测试策略也会不同。

#### (1) 程序安全测试要考虑的问题

- ◎ 明确区分系统中不同用户的权限。
- ◎ 在系统中会不会出现用户冲突。
- ◎ 在系统中会不会因用户权限的改变而造成混乱。
- ◎ 用户的登录密码是否可见、可复制。
- ◎ 是否可以通过绝对途径登录系统（通过复制用户登录后的链接进入系统）。
- ◎ 用户退出系统后是否删除了所有鉴权标记，是否可以使用后退键而不通过输入口令进入系统。

#### (2) 系统网络安全测试要考虑的问题

- ◎ 测试要采取的防范措施是否已正确装配好，有关的补丁是否已打上。
- ◎ 模拟非授权攻击，看防护系统是否坚固。
- ◎ 采用成熟的网络漏洞工具检查系统相关漏洞。
- ◎ 采用各种木马检查工具检查系统木马情况。
- ◎ 采用各种防外挂工具检查系统各组程序的外挂漏洞。





### (3) 数据库安全测试要考虑的问题

- ◎ 系统数据是否机密。
- ◎ 系统数据的完整性。
- ◎ 数据系统的独立性。
- ◎ 系统数据的可管理性。
- ◎ 系统数据可备份和恢复能力。

## 4.4.2 安全测试的目的

网站安全很容易被忽视，而黑客具备广泛的攻击手段，例如 SQL 注入、XSS、文件包含、目录遍历、参数篡改、认证攻击等，虽然我们配置了正确的防火墙和 WAF，但是这些安全防御软件仍然存在策略性的绕过问题。

我们需要定期扫描 Web 应用，但是手动检测所有的 Web 应用是否存在安全漏洞比较复杂和费时，所以就需要一款自动化的 Web 漏洞扫描工具，使用该工具可以：

- ◎ 提高 IT 产品的安全。
- ◎ 尽量在发布前找到安全问题予以修补，降低成本。
- ◎ 保障度量安全。
- ◎ 验证安装在系统内的保护机制在实际应用中是否对系统进行了保护，使之不被非法入侵，不受各种因素的干扰。

## 4.4.3 安全测试理论

安全测试理论如下：

- ◎ 安全测试强调完备性、覆盖性、可度量性。
- ◎ 安全测试强调系统本身的安全性，而不是外部防护系统的安全性。
- ◎ 安全测试提供的解决方案不是简单的防护，而是如何修复，进一步指导安全编程，最后辅助实施安全开发过程。



## 4.4.4 安全测试与功能测试的区别

安全测试与功能测试的区别如表 4-14 所示。

表 4-14

区别点	功能测试	安全测试
目标不同	以发现 Bug 为目标	以发现安全隐患为目标
假设条件不同	假设导致问题的数据是用户不小心造成的，接口一般只考虑用户界面	假设导致问题的数据是攻击者处心积虑构造的，则需要考虑所有可能的攻击途径
思考域不同	以系统所具有的功能为思考域	不但包括系统的功能，而且还有系统的机制、外部环境、应用与数据自身的安全风险与安全属性等
问题发现方式不同	以违反功能定义为判断依据	以违反权限与能力的约束为判断依据

## 4.4.5 安全测试与渗透测试的区别

安全测试与渗透测试的区别如表 4-15 所示。

表 4-15

区别点	渗透测试	安全测试
出发点	以成功入侵系统，证明系统存在安全问题为出发点	以发现所有系统的安全隐患为出发点
角度	站在攻击者的角度来看待和思考问题	站在防护者的角度来思考问题，尽量发现所有可能被攻击者利用的安全隐患，并指导其进行修复
覆盖性	只选择几个点作为测试的目标	在分析系统架构并找出系统所有可能的攻击界面后进行完备性测试
成本	无	需要对系统功能、系统所采用的技术及系统架构等进行分析，所以比渗透测试需要投入更多的时间和人力
解决方案	无法提供有针对性的解决方案	站在开发者的角度分析问题的成因，提供更有效的解决方案

## 4.4.6 安全测试工具介绍

Acunetix Web Vulnerability Scanner（简称 AWVS）是一款知名的 Web 网络漏洞扫描工具，它通过网络爬虫测试网站安全，检测流行的安全漏洞，如图 4-73 所示。



区块链：交易系统开发指南

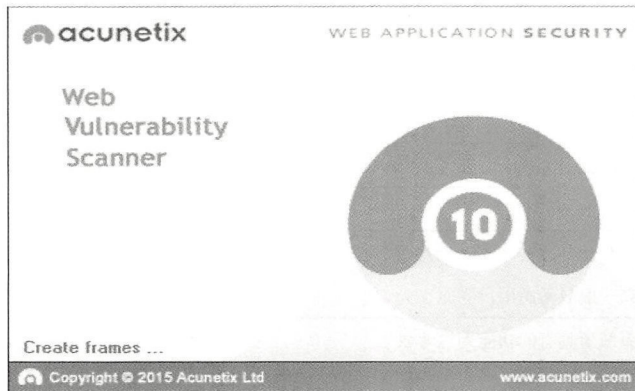


图 4-73

AWVS 安装成功后启动，程序界面如图 4-74 所示。

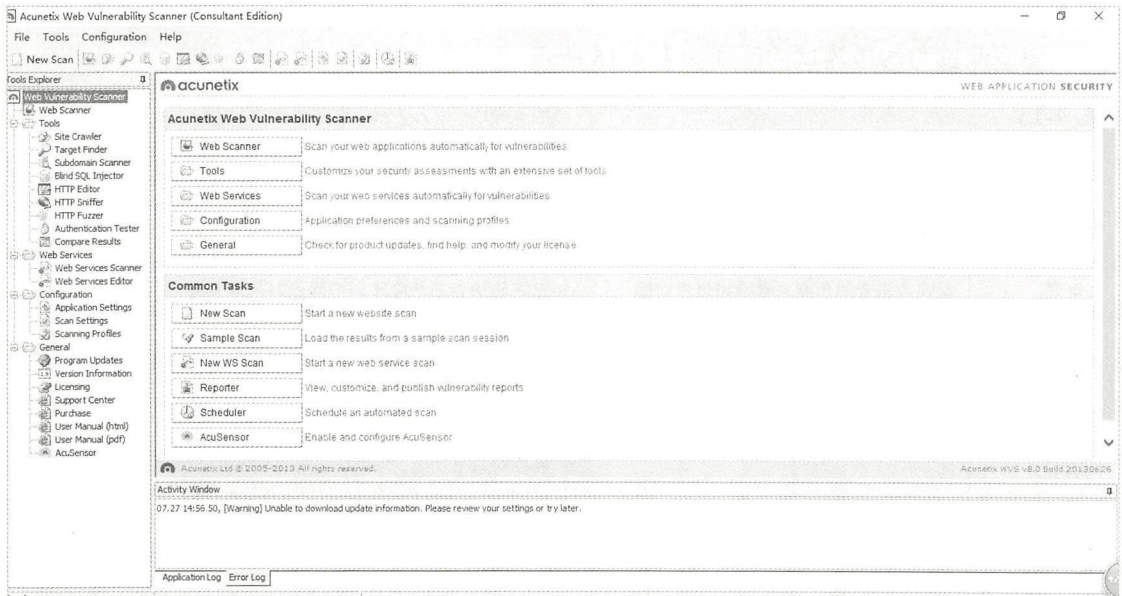


图 4-74

点击左上角的“New Scan”按钮，新建一次扫描，如图 4-75 所示。



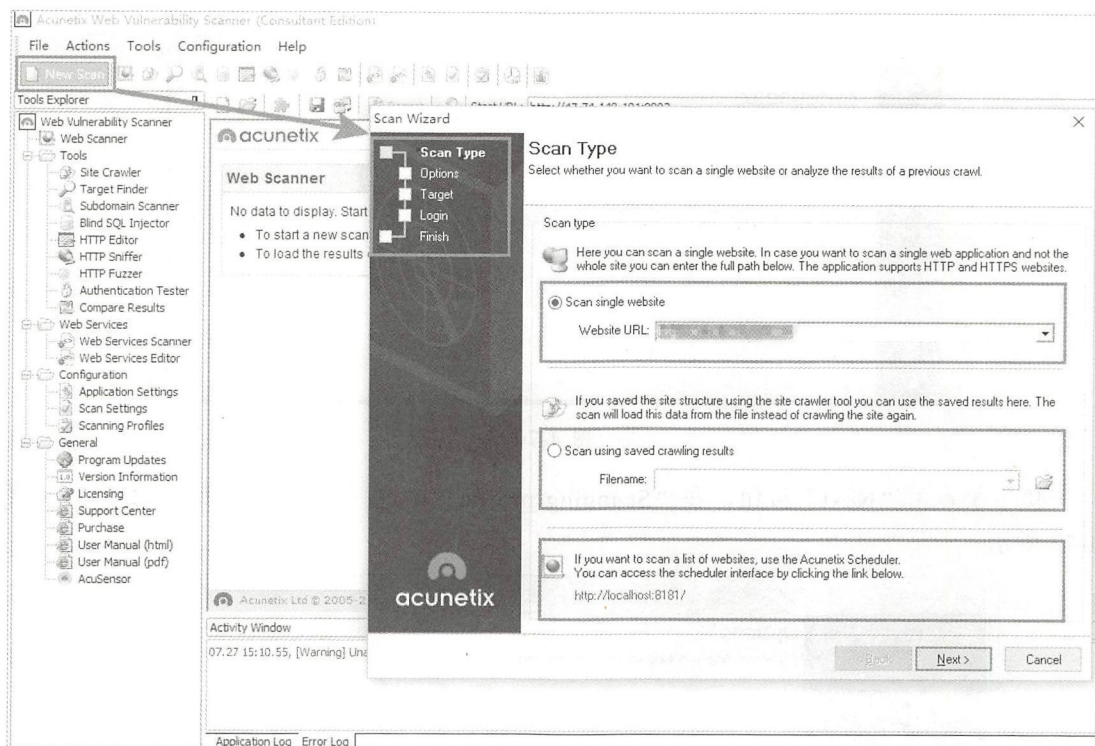


图 4-75

①Scan single website: 在“Website URL”处填入需要扫描的网站网址。如果想要扫描一个单独的应用程序,而不是整个网站,则可以填写完整路径。AWVS 支持 HTTP/HTTPS 网站扫描。

②Scan using saved crawling results: 导入 AWVS 内置的 Site Crawler 爬行的结果,然后对爬行的结果进行漏洞扫描。

③Access the scheduler interface: 如果被扫描的网站构成了一个列表(也就是要扫描多个网站的时候),则可以使用 AWVS 的 Scheduler 功能完成任务,访问所要扫描的网址,如图 4-76 所示。

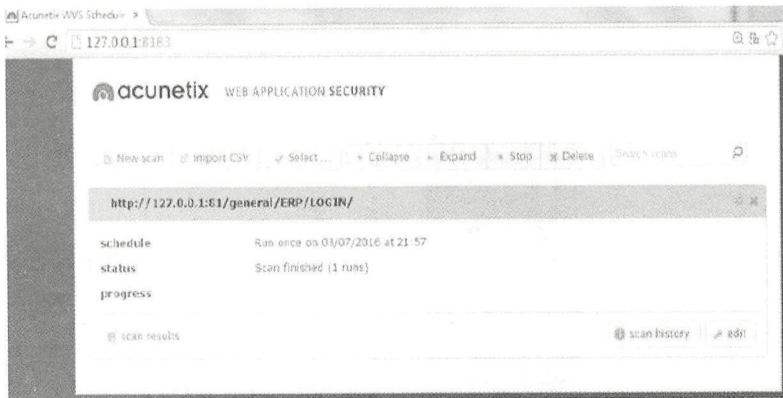


图 4-76

接下来点击“Next”按钮，在“Scanning profile”中选择安全扫描的漏洞类型，如图 4-77 所示。

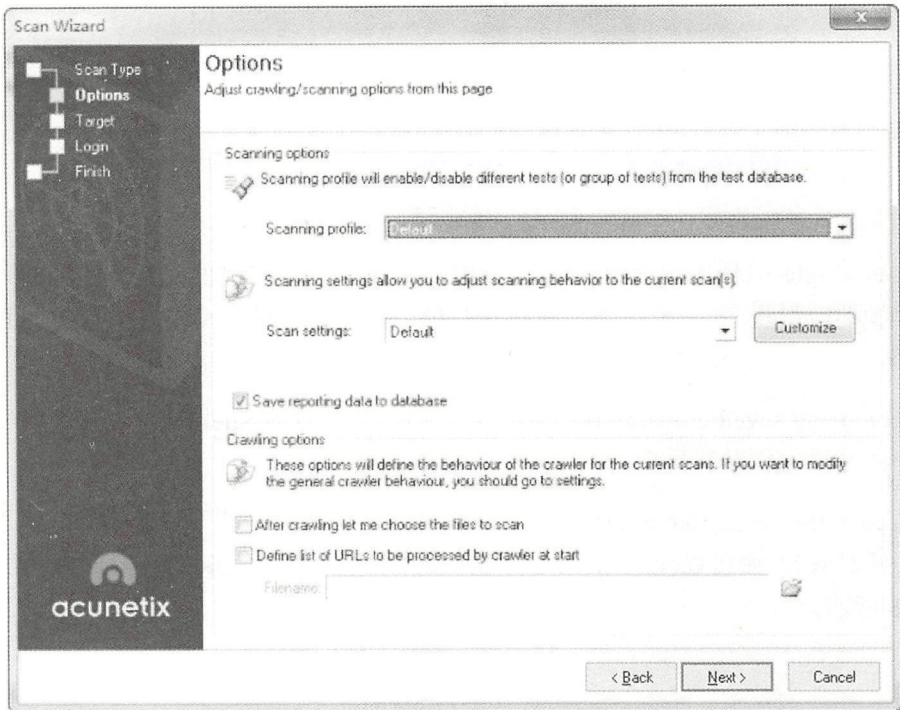


图 4-77

在“Scanning profile”中可供选择的漏洞类型如表 4-16 所示。



表 4-16

漏洞类型	说明
Default	默认设置，完全检测
AcuSensor	Acunetix 传感器机制，可提升漏洞审查能力，需要在网站上安装文件，目前主要针对 ASP.NET/PHP
Blind_SQL_Injection	SQL 盲注扫描
CSRF	检测跨站请求伪造
Directory_And_File_Checks	目录与文件检测
Empty	不使用任何检测
File_Upload	检测文件上传漏洞
GHDB	利用 Google hacking 数据库检测
High Risk Alerts	高风险警告
Network Scripts	网络脚本检测
Parameter Manipulation	参数操作
SQL Injection	SQL 注入检测
Test Search	文本搜索
Weak Passwords	弱口令
Web Applications	Web 应用程序
XSS	跨站请求检测

这里选择“Default”，点击“Next”按钮，AWVS 会自动帮助识别服务器的 Banner、OS 类型、Web 中间件、服务器脚本类型等信息，如图 4-78 所示。

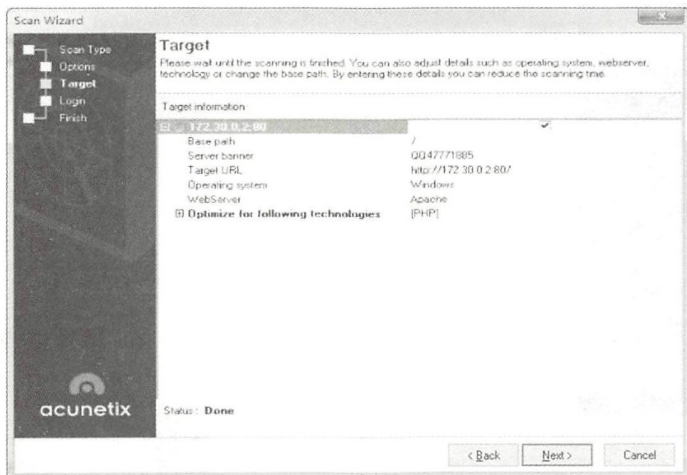


图 4-78





点击“Next”按钮，进入登录页面，如图 4-79 所示。

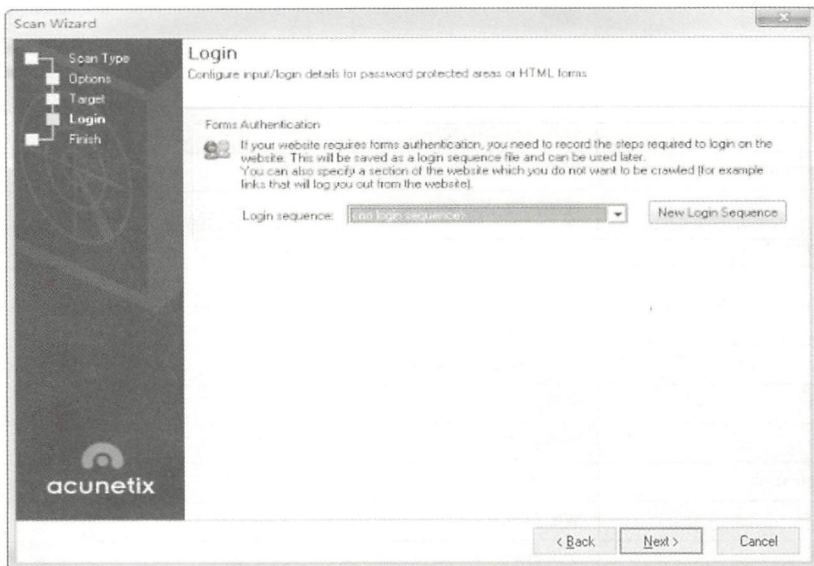


图 4-79

点击“Next”按钮，进入完成设置页面，如图 4-80 所示。

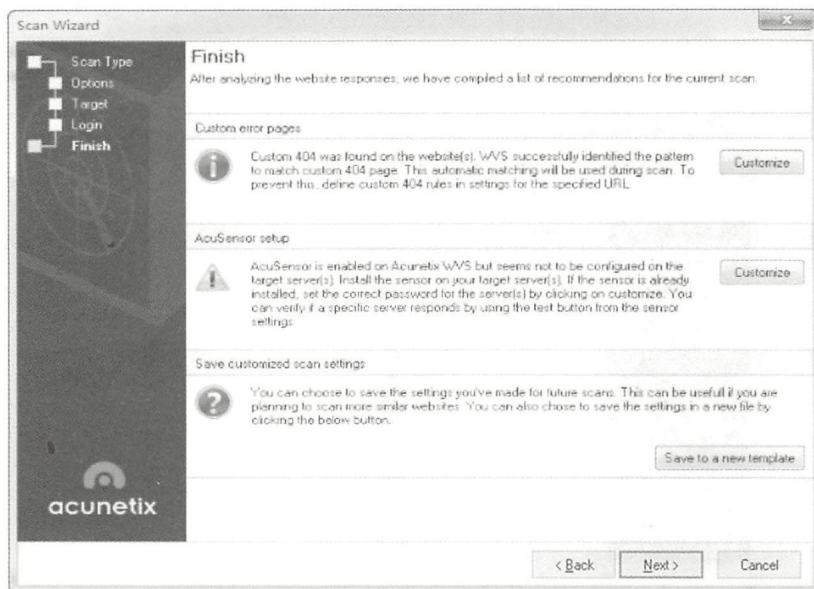


图 4-80

点击“Finish”按钮，完成 AWVS 的扫描设置。

现在进行安全扫描，扫描完成后结果如图 4-81 所示。

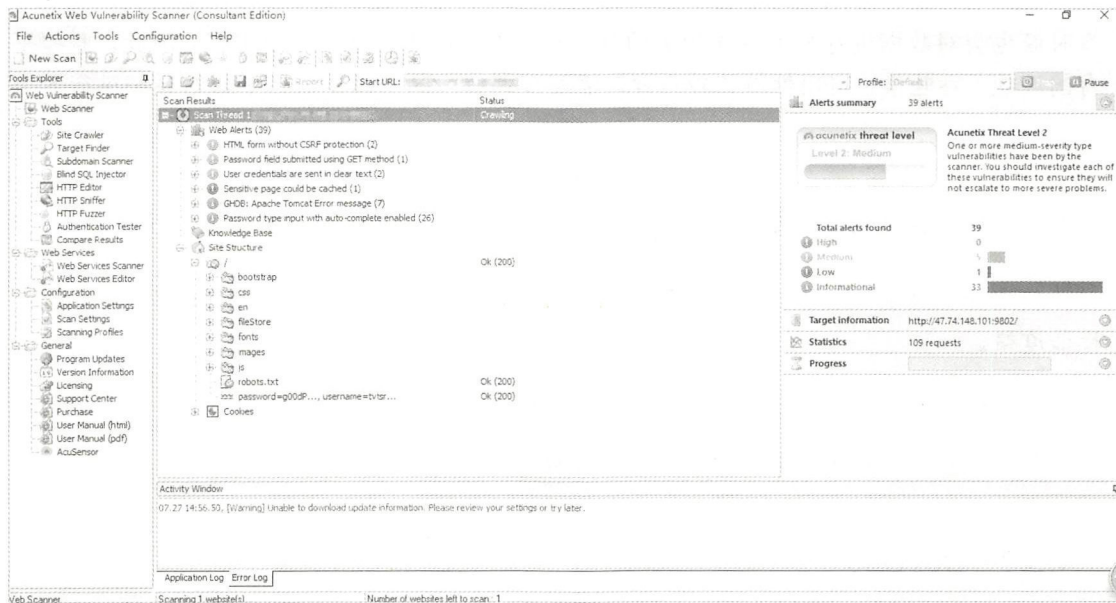


图 4-81

可以看到，右侧 Process 为 100.00%，说明安全扫描已经完毕。在“Total alerts found”下，红色的感叹号代表高危漏洞，黄色的感叹号代表中危漏洞，蓝色的感叹号代表低危漏洞，绿色的感叹号代表信息泄露。从扫描结果中我们就可以分析出是哪种类型的漏洞，从而进行修复。

## 4.5 系统运维

在运维方面，从硬件层面的 IDC 机房线路到防火墙等网络设计，都实现了自动化的主备切换能力，使用 Zabbix 完善了监控系统，除了对所有服务器和网络设备进行监控，还根据业务场景提供了数百个监控点，使我们可以在第一时间获得系统的运行状况和问题报告。运维和客服人员都是 24 小时待命的，确保不会因为管理空档期而带来意外故障，并为用户提供了随时可以联系报修故障的渠道，以便快速响应用户的问题。

大型分布式系统涉及各种设备，比如网络交换机、普通 PC、各种型号的网卡、硬盘、内存等，还有应用业务层次的监控，当数量非常多时，出现错误的概率就会变大，并且对有些监控的时效性要求比较高，如达到秒级别；在大量的数据流中需要过滤异常的数据，有时候也会对数据进行与上下文相关的复杂计算，进而决定是否需要告警。因此，监控平台的性能、吞吐量、可用性就很重要，需要规划统一的一体化监控平台对系统进行各个层次的监控。

### 4.5.1 平台的数据分类

应用业务级别：应用事件、业务日志、审计日志、请求日志、异常、请求业务 Metrics、性能度量。

系统级别：CPU、内存、磁盘、网络、I/O。

时效性要求：实时计算、近实时分钟计算、按小时/天/周/月的离线分析。

### 4.5.2 DevOps

DevOps 通过自动化的基础设施、自动化的工作流程和持续可测量的应用性能，来整合开发团队和运维团队，以达到更好的合作效率和生产率。DevOps 不仅仅是在开发者和运维者之间，更是一种文化的改变和鼓励沟通、交流、合作的行动，目的在于更加快速、稳定地构建高质量的应用系统，这恰好体现了精益管理的原则。如何达到 DevOps 呢？关键有两点：一是全局观，要从软件交付的全局出发，加强各角色之间的合作；二是自动化，人机交互就意味着手工操作，应选择那些支持脚本化、无需人机交互界面的强大管理工具，比如受版本控制的脚本，以及类似于 Zabbix 这样的基础设施监控工具和类似于 SaltStack、Ansible 这样的基础配置管理工具等。

DevOps 开发流程如下。

- ◎ 提交：工程师在本地测试程序后，将程序提交到版本控制系统如 Git 等中。
- ◎ 编译：持续整合系统（如 Jenkins CI），当检测到版本更新时，便自动从 Git 仓库拉取最新的程序进行编译、构建。
- ◎ 单元测试：Jenkins 完成编译、构建后，会自动执行指定的单元测试代码。



- ◎ 部署到测试环境：在完成单元测试后，Jenkins 可以将程序部署到与生产环境相近的测试环境中进行测试。
- ◎ 预生产测试：在预生产测试环境中，可以进行一些自动化测试，比如 Selenium 测试，以及与实际情况类似的测试，可由开发人员或客户手动进行。
- ◎ 部署到生产环境：当通过所有测试后，便可将最新的程序版本部署到实际生产环境中。

DevOps 开发流程图如图 4-82 所示。

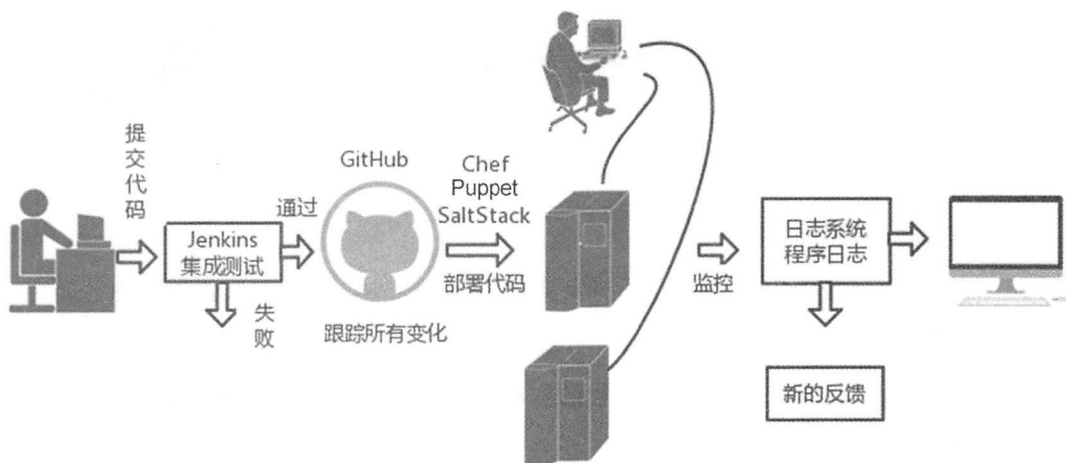


图 4-82

从图 4-82 中可以看到，借助 Jenkins 等工具，提交程序之后的步骤都可以自动完成，这节省了工程师的大量手动操作时间。由于每次提交程序之后都会自动进行编译与测试等流程，因此一旦有问题就可以马上发现并处理（Jenkins 会自动通知），这样也保证了程序的质量。

在图 4-82 中有一个很重要的角色：SaltStack。SaltStack 是一个架构管理系统，可以批量修改服务器的设定信息或执行命令，也可以通过配置管理使得开发环境、测试环境及应用环境尽可能保持一致。SaltStack 使得管理大量服务器变得更轻松，这方便了运维工作。与 SaltStack 类似的工具还有 Puppet、Chef、Ansible 等。

### 4.5.3 持续集成、持续交付、持续部署

#### 1. 持续集成

持续集成（CI）是指软件个人研发的部分向软件整体部分交付，频繁进行集成，以便更快地发现其中的错误。“持续集成”源自极限编程（XP），是 XP 最初的 12 种实践之一。

持续集成示意图如图 4-83 所示。

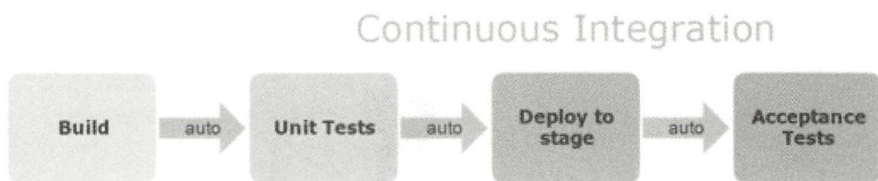


图 4-83

持续集成需要具备：

- ◎ 全面的自动化测试，这是实现持续集成和持续部署的基础，同时选择合适的自动化测试工具也极其重要。
- ◎ 灵活的基础设施，容器、虚拟机的存在让开发人员和 QA 人员不必大费周折。
- ◎ 版本控制工具，如 Git、CVS、SVN 等。
- ◎ 自动化构建和软件发布流程的工具，如 Jenkins、flow.ci。
- ◎ 反馈机制，如构建/测试失败，可以快速地反馈到相关负责人，尽快解决，以获得一个更稳定的版本。

持续集成的优点如下：

- ◎ “快速失败”，在对产品没有风险的情况下进行测试，并快速响应。
- ◎ 最大限度地减少风险，降低修复错误代码的成本。
- ◎ 将重复性的手工流程自动化，让工程师更加专注于代码。
- ◎ 保持频繁部署，快速生成可部署的软件。
- ◎ 提高项目的能见度，方便团队成员了解项目的进度和成熟度。

- ◎ 增强开发人员对软件产品的信心，帮助建立更好的工程师文化。

持续集成，该从何入手？

最重要的一环是有一个合适的持续集成系统。那么，是搭建私有部署系统还是选择托管型持续集成系统，关键在于团队运行的基础设施，以及团队对持续集成系统的资源投入力度。

对私有部署系统和托管型持续集成系统进行对比，或许能帮助你更好地做出选择。

- ◎ Self Hosted CI 指的是将软件部署在公司的机房或内网中，需要提供多台服务器来完成 CI 系统的运转，同时需要在不同机器之间进行环境配置。比如 Maven、Gradle 或 Jenkins，它们的特点是自由、开源，且文档支持广泛。其优点在于对构建环境有完全的控制权，能够实现完全定制；缺点是需要搭建环境和配置，维护成本高，需要买专门的机器，需要较多的人力、物力且更新迁移风险高。
- ◎ Hosted CI 指的是 SaaS 型的 CI 服务，全程在线进行构建配置，不需要考虑安装机器、安装软件、环境搭建等成本。常见的有 CircleCI、Codeship 和 TravisCI 等，以及国内的持续集成服务——flow.ci。SaaS 型的 CI 服务的特点在于无需额外的机器，几分钟就可以用起来，你可以根据需要动态调度资源，省时、省心、省力。

整体而言，过去 Jenkins 一直是大部分公司的选择，但现在随着公有云服务、Docker、SaaS 的普及，越来越多的企业开始选择 Hosted CI，也就是托管型持续集成系统。

另外，在选择合适的持续集成系统时，还需要考量系统的灵活度，以适应公司不同阶段的开发测试需求。

选择持续集成系统只是持续集成应用的其中一步，还需要建立合适的持续集成文化，比如代码质量管控、测试文化等。做好持续集成，为持续交付与持续部署打下坚实的基础。

## 2. 持续交付

持续交付在持续集成的基础上，将集成后的代码部署到更贴近真实运行环境的类生产环境（Production-like Environment）中。持续交付优先于整个产品生命周期的软件部署，建立在高水平的自动化持续集成之上。

持续交付示意图如图 4-84 所示。



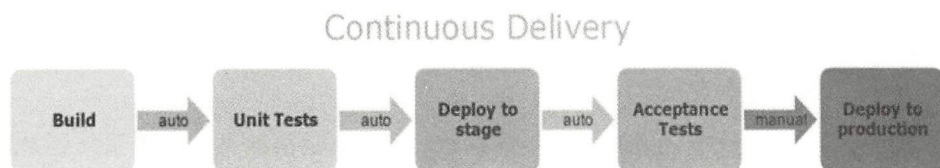


图 4-84

试想，如果等到所有东西都完成了才向下一个环节交付，将导致所有的问题只能在最后才爆发出来，解决成本巨大甚至无法解决问题。因此在完成单元测试后，可以把代码部署到连接数据库的 **Staging** 环境中进行更多的自动化测试。如果代码没有问题，则可以继续手动部署到生产环境中。当然，持续交付并不是指将软件的每一个改动都要尽快部署到生产环境中，而是指任何代码修改都可以在任何时候实施部署。

持续交付的优点如下：

- ◎ 快速发布，能够应对业务需求，更快地实现软件价值。
- ◎ 编码→测试→上线→交付的频繁迭代周期缩短，同时可以获得迅速反馈。
- ◎ 具有高质量的软件发布标准。整个交付过程标准化、可重复、可靠、进度可视化，方便团队人员了解项目成熟度。
- ◎ 具有更先进的团队协作方式。从需求分析、产品的用户体验到交互设计、开发、测试、运维等各角色密切协作，相比于传统的瀑布式软件团队，浪费更少。

### 3. 持续部署

持续部署是指当交付的代码通过评审之后自动部署到生产环境中。持续部署是持续交付的最高阶段，这意味着所有通过一系列自动化测试的改动都将自动部署到生产环境中。

持续部署示意图如图 4-85 所示。

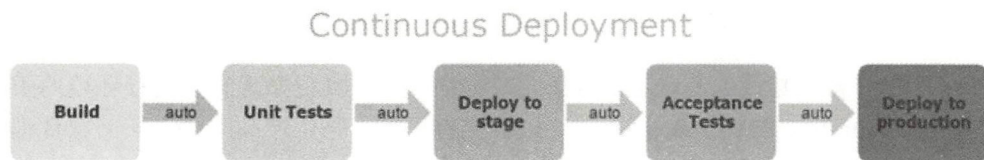


图 4-85

为什么说持续部署是理想的工作流程？

因为：开发人员提交代码，持续集成服务器获取代码，执行单元测试，根据测试结果

决定是否部署到预演环境中，如果成功部署到预演环境中，则进行整体验收测试；如果测试通过，则自动部署到生产环境中，全程自动化高效运转。

实际上，产品在从需求到部署的过程中会经过多种不同的环境，例如 QA 环境、各种自动化测试运行环境、生产环境等。这些环境的搭建、配置、管理，以及产品在不同环境中的具体部署，是比较复杂的，从头到尾全自动持续部署的确困难。但是如果能够实现持续交付，保证代码在模拟环境中没问题，那么团队成员就会做到真正的心中有数。

持续部署的主要优点是可以相对独立地部署新的功能，并能快速地收集真实用户的反馈。

#### 4. 持续集成和持续交付工具Jenkins

Jenkins 安装及配置如下。

##### (1) 安装

卸载自带的 Git（CentOS 6.5 自带的 Git 版本是 1.7.1）：

```
yum remove git
```

下载并编译安装 Jenkins：

```
wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo
rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key
yum install Jenkins
```

启动/停止 Jenkins 服务（默认端口是 8080）：

```
service jenkins start/stop/restart
```

##### (2) Maven 集成

在 Jenkins 的服务器上下载并解压 Maven 包，然后在“系统管理”→“Global Tool Configuration”中设置 Maven 的 Home，如图 4-86 所示。



图 4-86

### （3）Git 集成

先卸载 CentOS 默认安装的 Git 1.7 版本，再安装 Git。安装完成后，与 Git 相关的命令默认在 `/usr/local/git/bin/git` 目录下。

```
yum remove git
wget https://github.com/git/git/archive/v2.9.2.tar.gz
tar zxvf v2.9.2.tar.gz
cd git-2.9.2
make configure
./configure --prefix=/usr/local/git --with-iconv=/usr/local/libiconv
make all doc
make install install-doc install-html
```

接下来在“系统管理”→“Global Tool Configuration”中设置 Git 的 Home，如图 4-87 所示。



图 4-87

### （4）GitLab 集成

在 Jenkins 的服务器上创建一对 RSA Key，在 `~/.ssh` 下生成一对密钥 `id_rsa` 和 `id_rsa.pub`。

```
ssh-keygen -t rsa -C "username@qq.com"
# 一直按回车键，保持默认路径和文件名，不要密码
cd ~/.ssh
```

将公钥 `id_rsa.pub` 配置到 GitLab 服务器中，如图 4-88 所示。



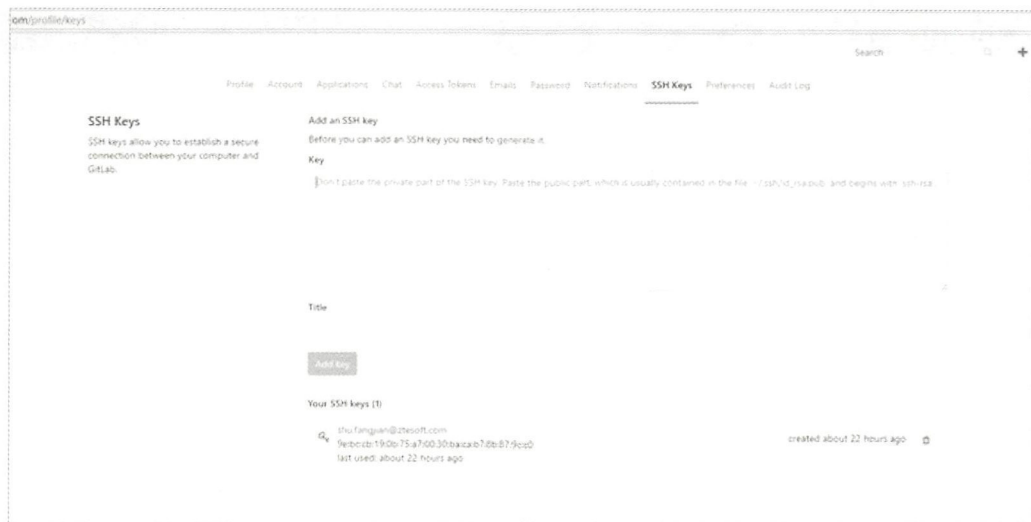


图 4-88

将私钥 id\_rsa 配置到 Jenkins 服务器中，如图 4-89、图 4-90、图 4-91 所示。

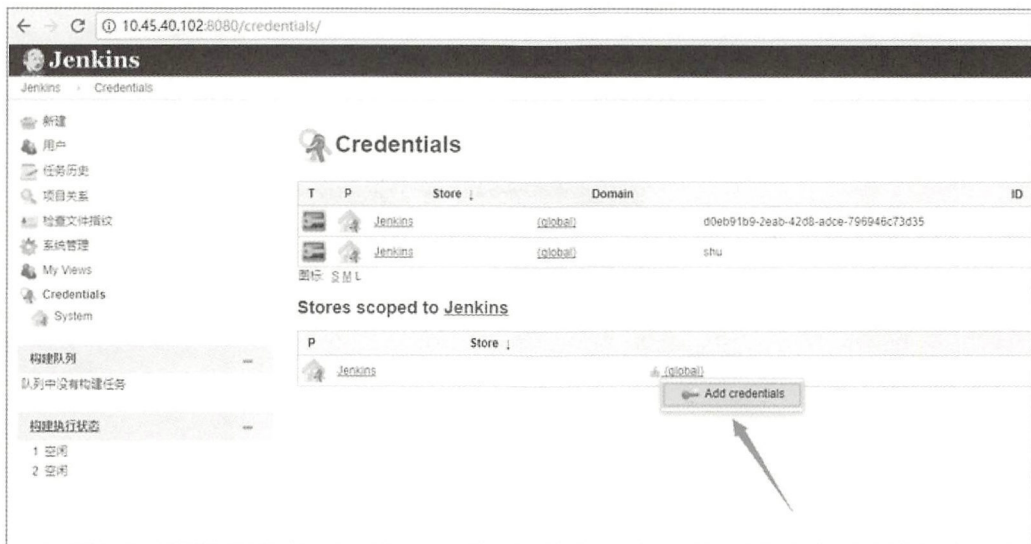


图 4-89

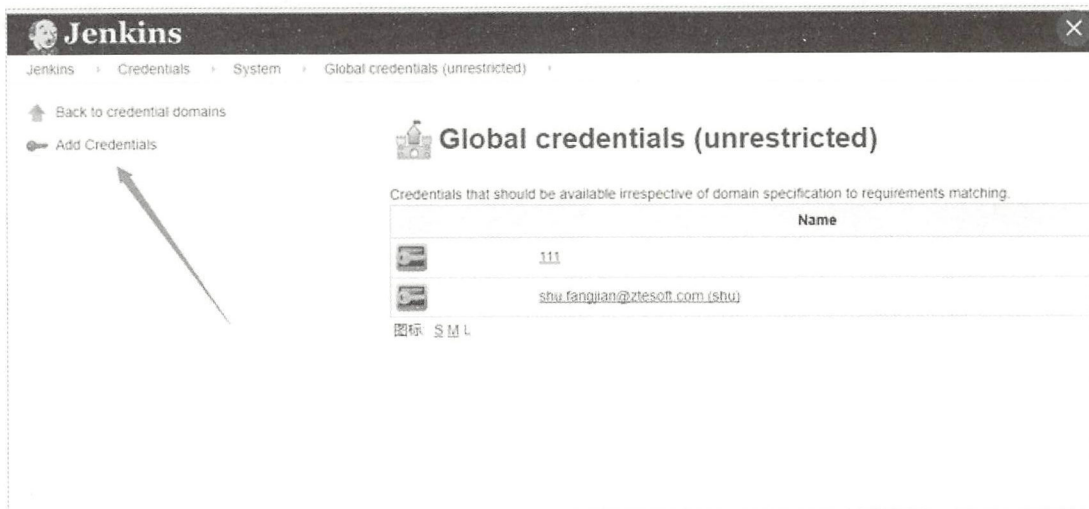


图 4-90

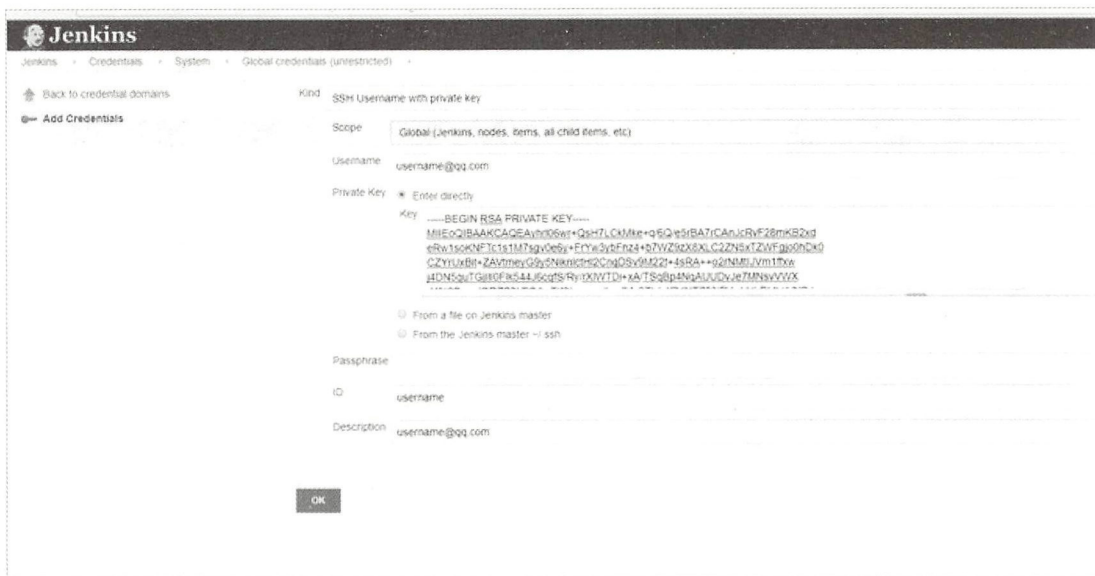


图 4-91

在“Username”中填写创建密钥时使用的用户名，在“Key”中填写 id\_rsa 的内容。

(5) 创建一个自动构建的 Job（Gitlab、Maven、Tomcat）

选择“构建一个 maven 项目”，如图 4-92 所示。

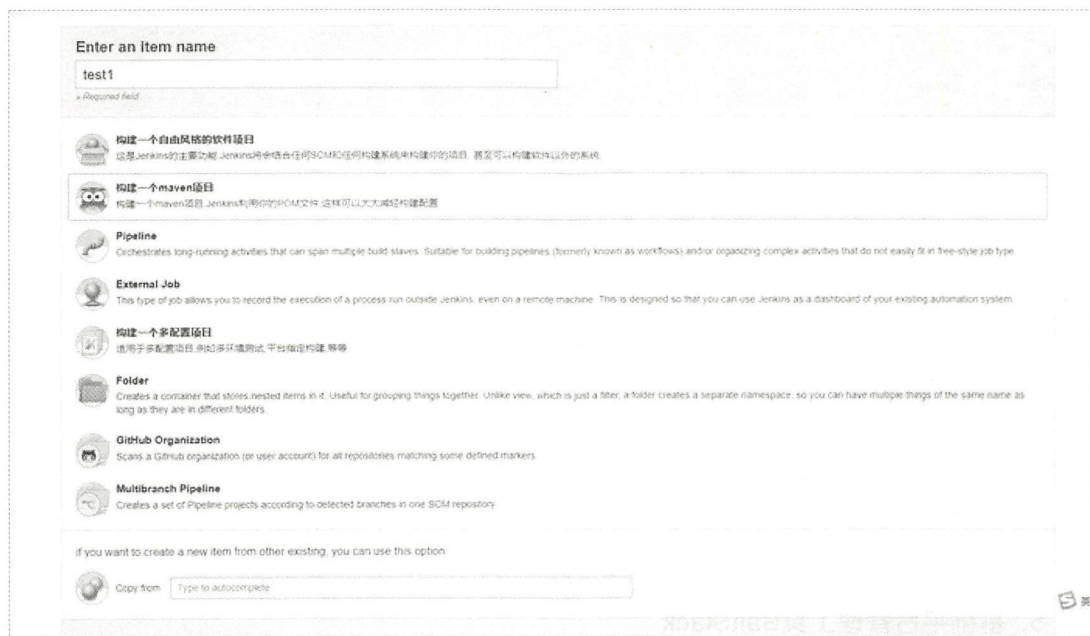


图 4-92

在“Credentials”中选择之前配置的私钥，如图 4-93 所示。

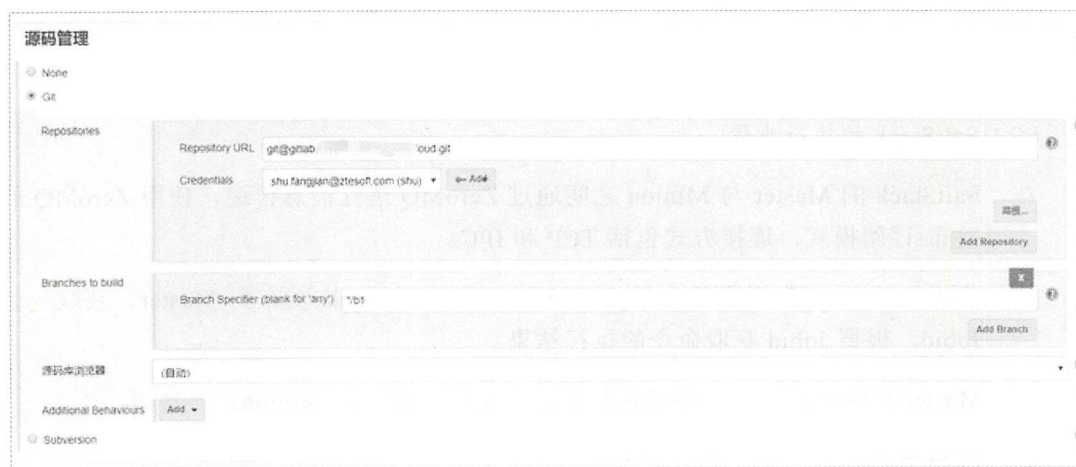


图 4-93

在“Goals and options”中填写 Maven 的指令，如图 4-94 所示。





图 4-94

构建完成后需要部署到 Tomcat 中，Tomcat 的用户名和密码必须在 tomcat-users 中配置好。

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<role rolename="manager-jmx"/>
<role rolename="manager-status"/>
<user username="tomcat" password="tomcat" roles="manager-gui,manager-script,
manager-jmx, manager-status"/>
```

## 5. 基础平台管理工具SaltStack

### (1) SaltStack 的基本原理

SaltStack 采用 C/S 模式，其服务器端是 Master，客户端是 Minion，Master 与 Minion 之间通过 ZeroMQ 消息队列通信，是一个同时对一组服务器远程执行命令和状态管理的工具。

### (2) SaltStack 的执行步骤

- ① SaltStack 的 Master 与 Minion 之间通过 ZeroMQ 进行消息传递，使用 ZeroMQ 的发布-订阅模式，连接方式包括 TCP 和 IPC。
- ② 将“cmd.run ls”命令从 salt.client.LocalClient.cmd\_cli 发布到 Master，获取一个 Jobid，根据 Jobid 获取命令的执行结果。
- ③ Master 接收到命令后，将要执行的命令发送给客户端 Minion。
- ④ Minion 从消息总线上接收到要处理的命令，交给 minion.\_handle\_aes 处理。
- ⑤ minion.\_handle\_aes 发起一个本地线程调用 cmdmod 执行 ls 命令，执行完后调用 minion.\_return\_put 方法，将执行结果通过消息总线返回给 Master。
- ⑥ Master 接收到客户端返回的结果，调用 master.\_handle\_aes 方法将结果写到文件中。

◎ salt.client.LocalClient.cmd\_cli 通过轮询获取 Job 的执行结果,并将结果输出到终端。

### (3) 安装 SaltStack

在 CentOS 6、RedHat 6 下安装:

```
sudo yum install https://repo.saltstack.com/yum/redhat/salt-repo-latest-1.el6.noarch.rpm
yum clean all
```

安装 salt-minion、salt-master 或其他组件:

```
sudo yum install salt-master
sudo yum install salt-minion
sudo yum install salt-syndic
sudo yum install salt-cloud
sudo yum install salt-api
```

在服务器端安装:

```
sudo yum install salt-master
```

在客户端安装:

```
sudo yum install salt-minion
```

### (4) 配置 SaltStack

在服务器端配置如下:

```
vim /etc/salt/master
# Master 消息发布端口, 默认为 4505
publish_prot:4505
# 工作线程数, 应答和接收 Minion, 默认为 5
worker_threads:100
# 客户端与服务器端通信的端口, 默认为 4506
ret_port:4506
# 自动接收所有的客户端
auto_accept:True
# 自动认证配置
autosign_file: /ect/salt/autosign.conf
```

在客户端配置如下:

```
vim /ect/salt/minion
```

```
# Master IP 地址或域名
master:10.0.0.1
# 客户端与服务器端通信的端口，默认为 4506
syndic_master_port:4506
# 建议线上用 ID 显示
id:test
```

被管理机器的 ID 是唯一的，不能重复，默认是被管理机构的 hostname；如果 ID 发生更改，则 Master 需要重新认证。

我们通过 tcpdump 做一个实验，在修改 Minion ID 后，在 Master 上会新增一个 ID，但旧 ID 还在，当执行 test.ping 时，执行时间变长了，延迟时间约为 14s，而且 Master 会在发送命令后，延迟 10s 再向每个已经执行成功的 Minion 发送一个包，且 Minion 有返回。如果没有 ID 存在，则不会发送，可以理解为 Master 在向每个 Minion 确认连接的 ID 信息，Minion 的 Salt 服务关闭也是这种情况，修改 timeout 值无效。

#### （5）测试 SaltStack

测试环境关闭 iptables：

```
Service iptables stop
```

在 Master 端执行：

```
Salt-key -L
Salt-key -A
```

查看到 Minion 端的 IP 地址，表示成功：

```
Accepted Keys:
192.168.1.3
DeniedKeys:
Unaccepted Keys:
Rejected Keys:
```

测试 ping 命令成功：

```
Salt '*' test.ping
192.168.1.3:
True
```



## 4.6 本章小结

本章以一个在线交易所为例，详细介绍了交易系统的前台及后台功能，包括从添加虚拟币到开启一个市场进行交易的过程、虚拟币在平台流转的过程、币币交易的撮合过程和手续费的收取方式，以及一些安全方面的知识等。读者通过本章的学习，一定会对交易系统的前台和后台功能有一个整体的认识，也一定有点跃跃欲试。他山之石，可以攻玉。本书只能为读者打开区块链交易的大门，若想走得更远，还需要不断地努力学习。

# 5

## 第 5 章

---

### 中心化区块链交易系统

区块链交易系统，顾名思义，即区块链数字货币的交易系统。当前交易系统按架构分为中心化区块链交易系统和去中心化区块链交易系统。

因为架构不同，这两种交易系统在业务逻辑上也有很大的不同，当然它们各有利弊，这两类架构会直接导致区块链交易系统的运行机制和操作流程不同。

下面将对这两类架构的交易系统的交易流程差异进行详细介绍。

#### 5.1 中心化区块链交易系统的特点

##### 5.1.1 中心化区块链交易系统的机制

中心化区块链交易系统和股票交易系统模式非常类似，其机制示意图如图 5-1 所示。

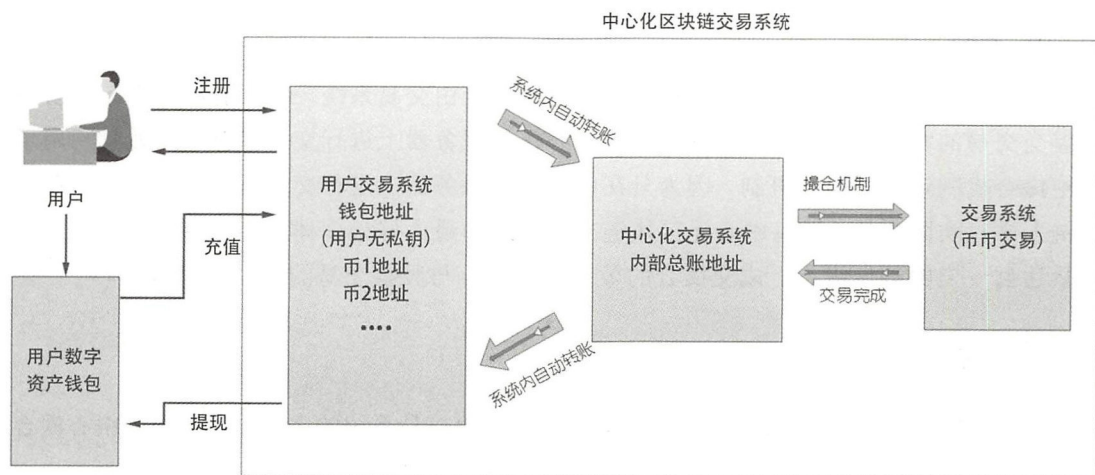


图 5-1

其主要流程说明如下：

### 1. 开户

用户通过中心化区块链交易系统提供的注册方法进行注册，注册时自动生成该用户在交易系统的地址集，可能存在多个地址，有些币隶属于不同的区块链，需要生成不同的公钥地址。此地址的私钥是由交易系统控制的，用户只有公钥地址。

### 2. 充值

用户使用自己的数字资产钱包向开户时生成的公钥地址转入需要交易的数字资产（另一种方式是交易系统依照币的属性，会分配对应的币的地址和通证对应的币的地址，供用户充值用于交易的数字资产）。

### 3. 自动转账

当用户从自己的钱包地址向交易系统的地址充值时，所有的数字资产会自动转入交易系统，完全由交易系统控制，用户采用挂出买单和卖单指令在交易系统中进行数字资产交易。

此处可以类比股票交易所，即用户把钱（币）交给交易所，由交易所作为信用背书代管用户的钱（币），这个时候钱（币）的实际控制人为交易所。



## 4. 交易

经过上面三步的操作，用户的数字资产已经完全由交易系统控制，用户只需向交易系统提交交易的相关指令，交易系统就会在中心化的服务器上进行交易撮合，此处就利用了集中撮合机制或中心撮合机制。因为只在中心化的服务器上进行交易操作，并未真正在区块链上进行实际交易，所以中心化区块链交易系统的最大好处就体现出来了，那就是当订单量达到一定的数量级后，成交撮合的效率非常高，与股票交易的实时性类似。

## 5. 提现

经过第四步的交易操作，用户的数字资产交易经由交易系统的中心化服务器的中心撮合机制完成，交易完成后的数字资产又回到用户指定的账户中。此时用户可以向交易系统提交提现指令，交易系统会通过中心表的操作将用户在交易系统账户中的余额做相应的减少，然后将所减少的这部分数字资产通过交易系统在区块链上的总账账户，向用户的区块链的钱包地址转账，即完成用户从交易系统的提现操作。

由上述步骤可知，所有的数字资产都在交易系统的地址中，所以提现的过程就是将数字资产由交易系统的地址转向用户钱包地址的过程。

### 5.1.2 中心化区块链交易系统的gas耗费

在中心化区块链交易系统的交易过程中，产生费用的地方有以下几个。

#### 1. 充值

当从用户的钱包地址向交易系统为用户分配的地址充值时，需要消耗 gas，费用由用户直接承担。

#### 2. 自动转账

当数字资产从交易系统分配给用户的地址自动转入交易系统的地址时，也需要消耗 gas，费用由交易系统直接承担。目前，主流的交易系统基本上均采用充值免费的模式，对用户来说这部分 gas 是无感知的。

### 3. 交易

在交易系统中进行交易时，交易系统会收取一定比例的手续费，一般是总交易金额的 0.1%（个别交易系统为 0.2%，使用交易系统代币打 5 折），费用由用户直接承担。不同的交易系统收取的手续费不同。

### 4. 提现

在进行提现操作时，从交易系统的地址转账到用户钱包地址，也需要消耗 gas，交易系统一般直接收取用户 0.01ETH 作为手续费。

## 5.1.3 中心化区块链交易系统的优劣势

### 1. 优势

中心化区块链交易系统在技术实现上有成熟的解决方案，即使面对海量的大并发实时交易，也依然可以为用户提供很好的服务，尤其是在交易失效性方面，其具有高并发、高可用和用户体验好等特点。

中心化区块链交易系统庞大的用户量和交易量也带来了足够的交易深度，提供了充分的流动性。

从经济上看，中心化区块链交易系统的交易成本是由市场环境和监管政策决定的，它可以根据运营策略来制定变化的手续费规则。为了鼓励用户高频交易，甚至可以不收取交易手续费，但一般都会对资产提现收取手续费。

由于中心化区块链交易系统的所有交易实质上都是 IOU 记账，因此从技术上看，交易成本是非常低的。

### 2. 劣势

#### （1）人为因素

中心化信用背书会面临内部运营风险、商业道德风险、资产盗用风险等，严重影响用户资产安全。有名的案例比比皆是，比如：

2013 年 10 月，比特币交易所 GBL 突然关闭，负责人卷款跑路，用户损失 2000 万美元的资产。

2014 年 2 月，Mt.Gox 监守自盗 85 万比特币，即大名鼎鼎的“门头沟事件”。

2014 年 5 月，FXBTC 长期亏损停止运营，疑似卷款跑路。

2015 年 1 月，加拿大交易平台 Virtex 停止提现，并将资金分批转走，疑似跑路。

2015 年 2 月 3 日，中国台湾比特币交易所 Yes-BTC 被盗，数百个比特币丢失，网站声明其董事长不知去向。

2016 年 4 月 7 日，交易所 ShapeShift 的钱包被黑客盗窃，后证实该盗窃行为属于监守自盗，黑客受一名离职员工的指使。

2017 年 1 月，比特币亚洲闪电交易中心卷款跑路，卷走上亿元的资金。

2017 年 7 月，BTC-e 交易所下线，随后域名被封禁，运营者 Alexander Vinnik 涉嫌洗钱、盗窃被捕。

2017 年 9 月 6 日，瑞通光泰投资基金管理有限公司旗下网站莱特中国卷款跑路，页面关闭，投资者被拉黑，资金不翼而飞。

2017 年 12 月 10 日，加密货币交易平台币集网网站关闭，疑似跑路，用户已报案维权。

## （2）技术因素

对资产的第三方背书集中式托管，会招致巨大的黑客攻击风险，相当考验网站技术能力和紧急应对能力。有名的案例也比比皆是，比如：

2014 年 3 月 1 日，美国加密货币交易所 Poloniex 被盗，损失占总量 12.3% 的比特币。

2014 年 8 月 15 日，加密货币交易所比特儿微博称被黑客盗走 5000 万个 NXT，价值 1000 多万元人民币。

2015 年 1 月 10 日，比特币交易所 Bitstamp 遭受黑客攻击，损失价值 510 万美元的比特币。

2016 年 5 月，中国香港数字交易所 Gatecoin 被盗 18 万个 ETH、250 个比特币。

2016 年 6 月，众筹项目 The Dao 因智能合约漏洞遭黑客攻击，被盗 360 万个 ETH。

2016 年 8 月，中国香港 Bitfinex 交易所由于网站出现安全漏洞，12 万个比特币被盗，当时价值 6500 万美元。

2017 年 7 月，韩国加密货币交易所 Bithumb 被盗，据评估损失达数十亿韩元。

2017 年 7 月，加密货币交易所 CoinDash 被黑客盗走大量以太坊，损失达 700 万美元。戏剧性的是，黑客分别于 2017 年 9 月与 2018 年 2 月，分两次将被盗以太坊原数返还。



2017 年 12 月 19 日，Youbit 第二次遭受黑客攻击，损失 17% 的数字资产，并宣告破产，用户补偿工作暂无后续进展。

2018 年 1 月 26 日，加密货币交易所 Coincheck 被黑客盗走大量 NEM，损失约 5.3 亿美元。

2018 年 2 月 10 日，意大利交易所 BitGrail 遭黑客攻击，损失 1700 万个 NANO 币，总价值约 1.7 亿美元。

2018 年 3 月 7 日，加密货币交易所 Binance 遭受黑客攻击，所幸没有出现丢币情况。

## 5.2 去中心化区块链交易系统的特点

去中心化区块链交易系统和中心化区块链交易系统有很多不同之处，但是最显著的区别人是，从用户直观感受上来说，通过以太坊的官方查询网站查询交易地址，中心化区块链交易系统只是一个 ERC20 地址，而去中心化区块链交易系统是智能合约地址。

下面将对去中心化区块链交易系统的运行机制进行详细说明。

### 5.2.1 去中心化区块链交易系统的机制

去中心化区块链交易系统与中心化区块链交易系统的本质区别就是用户有交易系统个人账户的私钥，可以全权控制交易系统账户。其机制示意图如图 5-2 所示。

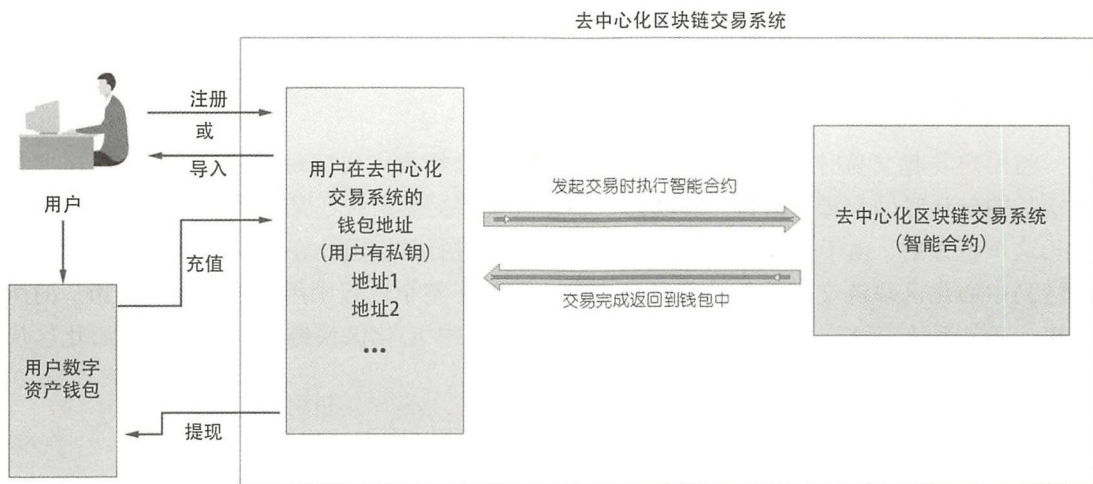


图 5-2

其主要流程说明如下：

## 1. 开户

通过交易系统平台注册用户交易账户，并设置密码。设置钱包地址有两种方式：第一种方式是直接通过交易系统注册钱包地址，并控制私钥，仅用户自己可以操作钱包地址的账户；第二种方式是通过导入已有钱包，然后通过交易系统进行币币交易。

值得注意的是，由于钱包是由用户直接控制的，如果密钥丢失，则交易系统无法找回密钥，资产丢失由用户自己负责。

## 2. 充值

目前，去中心化区块链交易系统主流基本上都是 ERC20 体系，所以充值过程比较简单，直接从钱包地址充值到交易系统的新地址（账户公钥）。如果是导入的钱包，则可以直接使用钱包中的资产进行交易。

## 3. 转账

理想的去中心化区块链交易系统没有自动转账这一步，自己的新地址就是交易主体，这是与中心化区块链交易系统的主要区别，当然，为了安全或者做其他考虑，有些去中心化区块链交易系统也会存在转账这个功能，这就要根据特定业务逻辑来定了，这里不再展开说明。

## 4. 交易

当用户发起交易时，直接执行交易系统的智能合约来完成交易，这个过程往往需要花费很长的确认时间和撮合时间，所以去中心化区块链交易系统在某种程度上是存在交易时间长这个特性的。由于所有的交易都是在区块链上进行的，因此在交易撮合的速度上是无法达到中心化区块链交易系统的交易速度的，但是在整个过程中用户一直拥有“币”的所有权，交易系统在这方面无任何掌控权。这一点是去中心化区块链交易系统与中心化区块链交易系统的显著区别。

## 5. 提现

提现的过程是指用户从交易系统的新地址（账户公钥）转账到自己钱包地址的过程。

如果是导入的钱包，则这一步可直接略过，当智能合约交易完成后就已经在提现了。

### 5.2.2 去中心化区块链交易系统的gas耗费

在去中心化区块链交易系统中一样会产生 gas 耗费，但是与中心化区块链交易系统略有不同，具体会产生 gas 耗费的地方有以下几个。

#### 1. 充值

从用户钱包地址充值到去中心化区块链交易系统的新地址(账户公钥)，需要消耗 gas，费用由用户直接承担。如果是导入的钱包，且钱包中已有资产，则不需要再进行充值操作，从而不需要消耗 gas。

#### 2. 自动转账

去中心化区块链交易系统无此步骤，故也无 gas 耗费(因业务需要存在自动转账功能，则 gas 耗费也存在)。

#### 3. 交易

去中心化区块链交易系统也会收取手续费，一般是总交易金额的 0.1% (个别交易系统可能费率不同，这与具体的交易系统的业务优惠相关)，费用由用户直接承担。

这里需要特别注意的是，在取消交易时也会产生 gas 消耗，因为整个过程类似于转账过程，所以会有 gas 的消耗。

#### 4. 提现

从交易系统地址充值到用户钱包地址，也需要消耗 gas，费用由用户直接承担。当然，如果提现账户就是导入的去中心化区块链系统的账户，那么 gas 是可以省掉的；但是如果转走的话，则同样需要 gas。

由以上几点可见，去中心化区块链交易系统只有交易手续费，其他过程均为区块链交易的 gas 消耗。



### 5.2.3 去中心化区块链交易系统的优劣势

#### 1. 优势

从业务视角来看，去中心化区块链交易系统的模式相对要简单些，它只需要承担主要的资产托管、交易撮合及资产清算，而不用考虑非区块链具体交易的功能，例如账户体系、KYC、法币兑换等。

用户在区块链上的账户公钥就是身份，不需要向交易系统注册个人信息，因此就不存在个人信息安全问题，也不需要 KYC。

去中心化区块链交易系统的一切都是通过智能合约来实现，将资产托管、交易撮合、资产清算都放在区块链上来执行和实现，无须担心黑幕操作，在区块链上一切都是透明的。

用智能合约来实现去中心化、去信任的交易机制，解决了人为干预因素产生的内部运营风险和商业道德风险，避免了如资产盗用等严重影响用户资产安全的风险。

用户的托管资产可以自由转移无需任何人审批，在安全上得到了足够的保障。另外，用户的账户密钥控制在自己手中，所以在技术上黑客攻击由集中式攻击交易系统改成针对分散的个人账户进行攻击，利润空间的下降反倒使相对安全性得到提升。攻击是否成功取决于用户的安全意识和习惯。当然，黑客也可以攻击交易系统造成系统瘫痪，或者更改服务器 DNS 来骗取用户的密钥，但这些攻击对于去中心化区块链交易系统来说造成的影响范围和程度会有所下降，考虑到时间成本和收益，黑客的获利空间并不大，这在某种意义上杜绝了被盗风险。

#### 2. 劣势

用户在去中心化区块链交易系统中的一切资产和交易操作都是以区块链交易来驱动的，因此在时效性上就会受到区块链本身确认速度的影响。目前，在以太坊上交易确认需要几十秒的时间，这对用户体验而言并不十分友好。

交易成本也会受到区块链本身交易费用的影响，因此对于小额交易而言交易成本会变得很高。而中心化区块链交易系统则不受此限制。

由于区块链网络交易处理性能低下，并不能处理大并发的实时交易，因此在交易量和交易深度上远远不如中心化区块链交易系统，在流动性上受到限制。

用户本身需要对账户公钥、私钥有足够的安全操作知识才能保障足够安全，否则账号遗失、被盗也是会经常发生的，而账号遗忘和丢失后资产很难找回。

## 5.3 本章小结

本章介绍了区块链交易系统的不同分类，以及各分类之间的差异，并从运行机制、操作流程上进行了详细说明，主要是从中心化区块链交易系统和去中心化区块链交易系统两个大类进行了具体的分析和阐述，论证了不同架构下的区块链交易系统的优势和劣势，以及在不同应用场景下的特点。

# 6

## 第 6 章

---

### 交易系统的演进

自从 2008 年比特币推出以来，区块链技术一直在飞速发展中，人们在不停地探索和尝试着各种区块链技术，希望拓展区块链的使用，提高性能和商业应用。

链币不分家。区块链的代币价值对人们了解、关注、采纳区块链技术起到了至关重要的作用。它正在努力为当前存在的支付系统提供更有效的方案，同时吸引着众多资本及技术研发人员去尝试、创造、发展一个更快速、更安全、更透明的数字货币交易系统。

#### 6.1 去中心化

---

有一位区块链爱好者 Peserlin，他有一些 ETH，他想要用 ETH 去买 SWTC，但是 SWTC 只在 A 交易所才能买到，而 A 交易所并没有 ETH，并且只能通过 BTC 去买 SWTC。

这是在现实世界中可能会遇到的一个场景。那么，Peserlin 想要成功用 ETH 买到



SWTC，需要经历以下几个环节。

- ◎ 从自己钱包将 ETH 转账到另一个交易所 B。
- ◎ 在 B 交易所通过币币撮合交易消费 ETH 购买 BTC。
- ◎ 从 B 交易所将 BTC 转账到 A 交易所。
- ◎ 在 A 交易所通过币币撮合交易消费 BTC 购买 SWTC。
- ◎ 从 A 交易所将 SWTC 转账到自己钱包。

上述每一笔基于区块链的转账都要消耗 gas 作为手续费，gas 的比例不等。多一笔转账操作就会多一笔手续费，多一次等待到账时间。随着交易量的激增，网络会越发拥堵，想要快速交易，就只能通过提高 gas 的方式，这样交易成本也就涨上去了。

原本一次交易就可以完成的事情，一定要这样大费周折，兜个圈子才能实现吗？目前在大多数交易所中只能这样，因为它们是中心化交易所。

未来的交易所形态到底是中心化的（CEX，Centralized Exchange）还是去中心化的（DEX，Decentralized Exchange），争论一直没有停歇。

### 6.1.1 中心化交易系统

中心化交易系统一般都提供账户体系、KYC（Know Your Client）审核、资产充值、资产托管、交易撮合、资产清算、资产兑换等业务模块，在技术实现上已经有相对成熟的解决方案。

中心化交易所数字资产的流转示意图如图 6-1 所示。

- ① 用户在交易所外部发起充币请求，链上交易。
- ② 交易所监测区块链交易并在链上确认①交易金额到账用户热钱包后，自动发起链上转账到交易所冷钱包的交易。
  - ① 交易所发起由交易所冷钱包向交易所热钱包的链上转账交易。
  - ② 交易所监测区块链交易并在链上确认①交易金额到账交易所热钱包后，用户发起的数字资产转出操作会自动生成由交易所热钱包向用户提币冷钱包的链上转账交易。

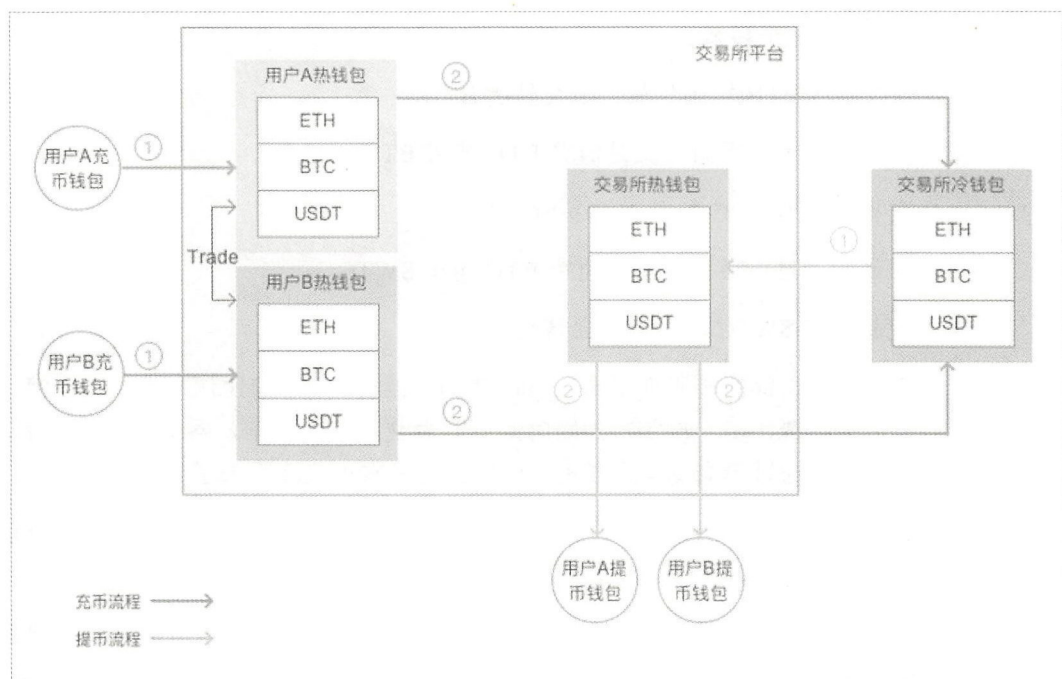


图 6-1

**Trade:** 交易所内部的币币撮合交易，各币种之间的资产交换只是数据库中值的变化，并没有链上操作，无 gas，无须等待到账，高效、快速，交易成本低。

但是中心化交易系统也存在着很多弊端和隐患，列举如下。

### （1）应对突发情况能力弱

区块链市场多变，机遇稍纵即逝，在市场需求的追赶下，紧急拼凑的产品难有完善的保障策略及应急预案，在面对突发状况时往往没有很好的应对措施，比如防篡改性较差，甚至搞个优惠活动都能因为流量突增造成服务器宕机。

### （2）安全事件频出

区块链领域早已成为黑客的众矢之的。从早期的 Mt.Gox 被盗事件，到 Bitfinex、OKEX、币安、火币等爆出的安全事件，在损害用户利益和交易所声誉的同时，也对整个行业造成震荡性影响。用户将数字资产存放在这些交易所中，不仅要面对交易所防盗措施不到位的风险，还要为交易所“监守自盗”的可能性担惊受怕，甚至包括政府的禁止访问风险。

### （3）割裂流动性，价格可操控

从整个市场层面来看，主流的中心化交易系统对全球的数字货币流动性进行分割，但在交易量上没有绝对的市场领先者，中心化交易系统把控某些项目是否上交易所和项目的定价，可能会因为信息、资金流动不充分而导致价格操纵行为的产生。

### （4）交易信息不透明

中心化交易平台大多采取 IOU（I Own You）记账方式，实际上这是一种抵押债券的方式。用户将资产转入交易所的账户后，他们的账户将记录一笔相应数字的欠条（IOU），而真正的资产早已转移到交易所的冷钱包中了。交易所内的资产交换只是数据库中数据条目的变化，在区块链网络中查不到任何记录。所有数据都由交易所完全掌握，只有当用户发出提现请求时，他们才会收到实际的资产。

在这种模式下，用户资产是否被合理使用，交易所运营方是否有虚假交易，充值后或提现后是否能及时到账？在没有监管的环境下，交易所只能凭着自己的良心对用户的数据负责。重要的交易信息被中心化占有，这本身也是不符合区块链去中心化的本意的。

### （5）KYC 审核

KYC 是一把“双刃剑”，在追查黑客和不法分子时，KYC 预留的信息可以提供很好的帮助，但是大量的交易者都是在做合法交易的，中心化交易平台通过 KYC 收集用户信息，对于这些合法交易者来说就是一种潜在的信息安全隐患。我们无法确保交易系统不会利用这些个人信息作恶。

### （6）用户的信任

对于区块链资产交易，用户关注的是：

- ◎ 个人资产安全。
- ◎ 个人信息安全。
- ◎ 交易便捷（提币限制少、转账到账快）。
- ◎ 交易费用低。
- ◎ 平台流动性强（币种丰富，交易量、交易深度、提币限制少）。
- ◎ 支持多种衍生品。
- ◎ 可提供 API 接口。



其中安全、交易成本和体验是最受关注的几个方面。可往往鱼与熊掌不可兼得，目前高人氣的中心化交易平台往往只能做到其中的一个方面或者两个方面，并且伴随着较高的交易费用。

### 6.1.2 去中心化交易系统

与中心化交易系统相比，去中心化交易系统的工作流程截然不同。

以 Ethereum 为例，去中心化交易系统由两部分组成，第一部分是智能合约。这个智能合约可以与 Ethereum 区块链中支持的 ERC20 代币进行交互，而且是实际交易的执行者。第二部分是去中心化的订单簿和匹配的供应商。用户可以向供应商提交交易请求，但不释放实际的代币。如果供应商可以找到匹配的对象或匹配路径，它就将交易请求提交给智能合约，智能合约则执行原子性交易，用户的代币只在这个原子性交易中予以传输。

去中心化交易系统有多种好处，主要体现在：

- ◎ 避免任何资产被托管。这是去中心化交易系统的核心优势。在实际交易发生之前，用户完全拥有自己资产的处置权，不由任何机构代为持有。用户只会发布其期望交易的价格、数量、交换和被交换的货币，链上撮合找到匹配的对家后，完成原子性交易。由智能合约保证原子性交易的安全性，大大降低了用户对交易系统的信任门槛。
- ◎ 数字货币的安全性仅仅依赖用户自己的安全措施。只要用户以很好的方式保存自己的钱包密钥，就可以大幅度降低代币被窃取的风险。因为这种方式使得针对个人用户的黑客攻击无利可图。
- ◎ 用户和交易所之间没有所谓的往来转账，不产生额外的费用，也不需要额外的等待到账时间。在去中心化的交易环境中没有所谓的充值和提现的需要，也不存在所谓的代币从交易所充值、提现的事情，简化了操作，提升了体验，同时避免了在等待到账时行情波动造成的价值风险。
- ◎ 链上撮合，解决了交易所割裂流量的问题，支持多个交易所的交易请求在同一个交易池中完成撮合，增强了资产的流动性，抹平了交易所之间的价格差，增加了中心化交易所幕后操作价格走势的难度。
- ◎ 由于去中心化交易系统采用去中心化的方式工作，它可以与区块链的其他去中心

化的功能集成在一起，比如智能合约。此外，这种机制可以跨国家和地区部署，没有某一领域的管辖权要求。

从技术上讲，井通公有链首创的跨链功能和异步智能合约很好地支持了不同链之间的价值交换情况。

### (1) 井通区块链分层架构

井通区块链分层架构示意图如图 6-2 所示。

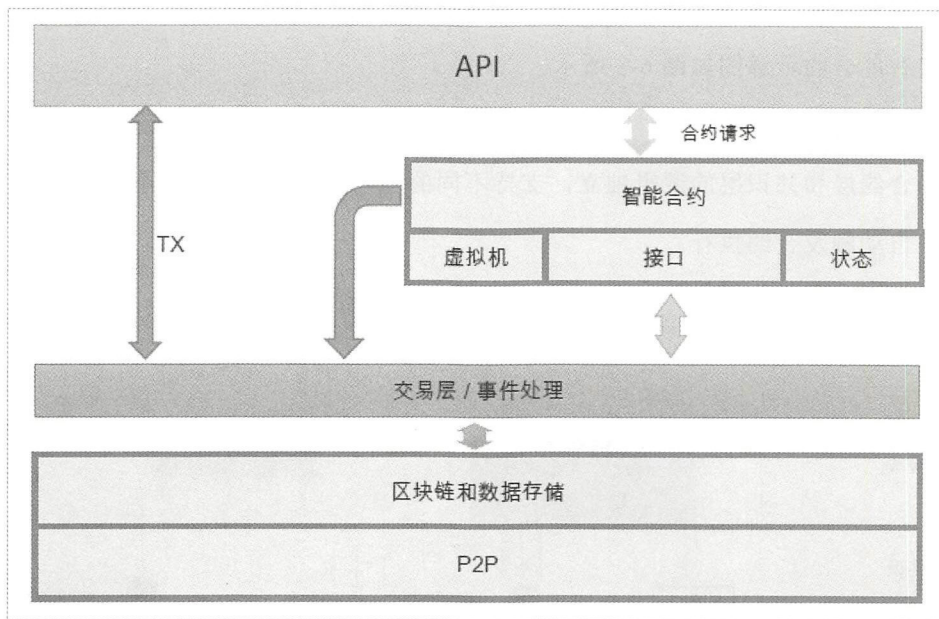


图 6-2

- ◎ P2P 层：该层定义了 P2P 协议。
- ◎ 区块链和数据存储层：该层处理与区块链操作相关的所有操作，如共识、数据访问等。
- ◎ 交易层：该层处理 TX 请求和回复，还处理控制流 TX 请求，并在必要时调用与智能合约相关的操作。
- ◎ 智能合约层：该层在虚拟机内部执行智能合约的程序，也保持临时合约状态。
- ◎ API 层：该层处理最终用户输入并从下一层获取输出。

## （2）跨链功能

- ◎ 主从链的跨链功能，实现资产在不同链之间的流动。
- ◎ 基于影子资产的跨链联动，实现从链资产和主链资产的交互。
- ◎ 自定义主链、从链资产交换协议。
- ◎ 实现主链和从链资产的价值互动，避免侧链等技术的价值冻结。

## （3）异步智能合约

异步智能合约示意图如图 6-3 所示。

- ◎ 异步调用合约机制，支持复杂的裸机合约，合约模块不会阻塞系统公示和其他模块。
- ◎ 合约层和共识层的逻辑独立，支持不同的合约虚拟机，实现模块化和热插拔。
- ◎ 自动触发合约执行。

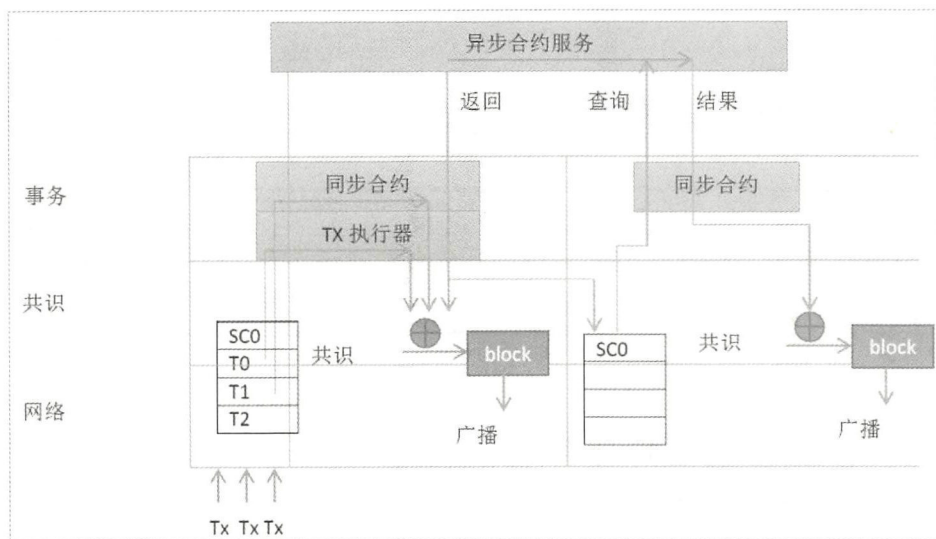


图 6-3

## 6.2 证券化

数字货币源于金融领域，区块链技术解决了数字货币的安全问题，促进了数字货币的



出现和发展。除对金融领域的影响与日俱增以外，数字货币也将给社会带来革命性的变化。数字货币作为价值传递的媒介，使得很多传统行业行规都会在数字货币浪潮的冲击下做出改变。各国政府对数字货币的态度是冷热不均，但唯一达成共识的是对数字货币交易进行规范化。有来自学者、官员、从业者的很多声音都支持对数字货币交易所实施更严格的规定，以遏制非法使用数字货币进行洗钱等金融犯罪和资助恐怖主义的行为。

数字货币及其交易平台的出现，使得企业有了自主发行企业数字货币（企业币）的渠道。利用企业币直接对接企业自身的实物产品或者服务，可以直接购买企业的产品，并且这些数字货币也可以在交易平台上进行交易。企业币的价格会随着企业的运营状况、行业的发展状况、国家政策等上下波动，这些都是影响企业币价格的重要因素。

2018 年 3 月 7 日，美国证券交易委员会（SEC）发布了《关于可能违法的数字资产交易平台的声明》，声明中称，依照联邦法律的标准，在虚拟货币交易所交易的资产属于证券范畴，因此相关交易所需要在 SEC 注册或获得注册豁免。

如何以合规的证券模式进行数字货币交易，是数字货币交易系统的下一个前进方向。

## 6.3 本章小结

本章列举了中心化交易系统的弊端，介绍了去中心化交易系统的优势，并对证券化发展方向进行了展望。

# 7

## 第 7 章

---

### 总结

本书从区块链的概念开始，讲述了区块链交易系统是怎么产生的、开发技术的难点，以及各种链的对接方案等。

数字货币交易与证券交易、期货交易的最大区别是交易连续不间断，波动没有涨跌幅限制，开发难度在于系统的稳定性和高并发性、高可靠性。

### 7.1 完美支持各种链

---

一个交易系统要想有更好的体验和更多的使用者，就需要支持各种链，以及这些链上的各种代币。本书主要讲解了 BTC、ETH、SWT、MOAC、EOS 公有链的特点和 API 调用，读者通过阅读这部分内容将对公有链有一个基本的认识。

比特币是最早的区块链应用，作为区块链 1.0 时代，没有账户的概念，用户余额是从

各自在区块链上所有未花费交易输出（UTXO）计算得来的。采用 PoW 共识机制，依赖机器进行哈希运算来获取记账权，无法避免矿池算力集中的问题。比特币没有智能合约，除了比特币，没有其他的数字货币，节点众多且分布广，达成共识的时间非常长。其开发的难点在于如何让用户在无感知的情况下接受长时间的成交等待时间。唯一的解决办法就是用中心化交易代替去中心化交易。

以太坊可以被解释为区块链+智能合约。其具备图灵完备性，支持智能合约，可以实现各种商业与非商业环境下的复杂逻辑，隐藏了底层技术的复杂性，让应用开发者更多地专注在应用逻辑及商业逻辑上。以太坊的不足之处在于其扩展性较差，和比特币一样遭受着每笔交易都需要网络中的每个节点来处理这一困境的折磨。2000TPS 的交易就可能导致以太坊链上的存储快速增长而拥堵。随着应用接入的增多，后期可能更加拥堵。好在以太坊全节点只需存储状态而不是完整的区块链。

在数字资产实现上，目前比特币系统无解，不能在它的上面构建数字资产；以太坊则走入了“唯智能合约论”，完全忽视智能资产无差别采取合约的 overhead。Ripple 采取银关 Fingate 模式，引入法币。

井通则是双向路线，采用银关模式管理简单的智能资产，在这里实现万物相连，万值相易，并且用类 OSPF 协议打造了高效的撮合算法。与此同时，井通也支持智能合约，对复杂的智能资产进行管理。采用这样的处理方式，是因为智能合约有 overhead，对于简单的智能资产是没必要浪费智能合约的功能的。

MOAC 公有链的创新之处在于智能合约的异步调用和分片处理，该项目旨在提供一种可扩展且有弹性的区块链，支持基于分层结构的状态交易、数据访问和控制流程。它创建了一个框架以允许用户采用高效的方式执行智能合约。它还提供了开放的体系结构，采用底层基础设施来快速、简便地产生子区块链。

EOS 作为叫板以太坊的公有链，最有技术特点的地方在于可以简化用户账号的生成与管理，并且能恢复账号，这在用户看来提高了安全性，并且号称支持百万级 TPS 的交易速度也让其赚足了公众的眼球，完全零费率，并可以快速且容易地部署去中心化应用。

本书对以上几种链进行了分析，并给出了调用这些链的接口的方法。读者可以以此为基础更深入地研究这些链，并对它们进行更广泛的应用。



## 7.2 稳定、高可用的系统

从交易系统 7×24 小时不间断交易和无涨跌幅限制的特点出发，设计出的系统必须是稳定、高可用的。

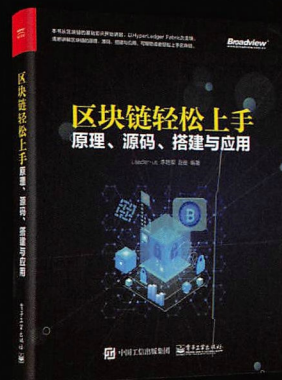
本书讲到的系统架构正是从如下几个方面出发进行设计的。

- ◎ 以空间换时间。
- ◎ 并行与分布式计算。
- ◎ 多维度可用。
- ◎ 伸缩性。
- ◎ 优化资源的利用。

## 7.3 交易系统功能齐全

一个完整的交易系统必须有用户注册、登录、实名认证、充值、提币、挂单、撤单、交易记录查询等功能，并提供完善的管理链、币、交易价格及交易手续费设置等一整套后台管理功能。

本书详细介绍了交易系统前台和后台功能开发的细节，包括大量的数据库设计、核心代码及流程图、效果图等，为交易系统的开发者提供了完整而又翔实的第一手资料。希望读者能够从本书中获得开发交易系统的真谛，并在本书所讲内容的基础上开发出更加完美、好用的交易系统。



# 区块链交易系统开发指南

区块链正在步入3.0时代，越来越多的传统企业与区块链结合，发行自己的数字资产，交易系统在数字资产的交易中起着重要的作用。通过阅读本书，开发团队可以快速开发出一个属于自己的区块链交易系统，让数字资产在一定范围内自由交易，形成相对稳定的企业联盟。本书内容通俗易懂，通过大量的源码和示例介绍区块链交易系统的开发过程，是一本难得的讲解区块链技术重要环节的好书，值得区块链开发者、爱好者和区块链技术公司人员阅读。本书一定会在区块链的发展历程中留下光辉的一笔。

程晓明

中国社会科学院博士、新三板教父

交易是一种基本的经济活动，在区块链生态中其集中在交易所是不合理和不科学的，交易应该随时都能得到实施。从这个角度来说，交易系统是生态的基础设施。本书从技术角度系统阐述了交易系统，不仅可以为技术人员提供参考，也可以为相应的监管部门提供参考。

高斌

中国区块链普及工程推进联盟秘书长、旗点商学院院长

传统的证券交易市场通常赋有国家荣誉属性，其核心交易系统大多是由精英团队开发的，以保障高吞吐量、高安全性的交易。交易系统的架构与技术往往不被外界所知，除在专业性很强的期刊或专利搜索上可以查询到部分内容以外，系统性介绍相关内容的书籍基本没有。近年来，随着比特币等数字资产的蓬勃兴起，相关的交易需求日益增加，交易系统的建设也呈现出社区化与平民化的趋势。本书所讲解的数字货币的交易系统技术基本上是以开源软件为基础的，加上钱包模块等处理加密货币必要的组件，全书系统性、逻辑性和实用性都非常强，是人人可以写数字货币交易系统时代不可多得的发展指南。从完备性的角度来看，没有理由认为基于本书开发出来的系统，在可用性和安全性上不可以比肩传统证券交易所的核心系统。

左涛

中国香港交易所原董事总经理兼HKEX内地技术公司总经理、大连商品交易所原首席架构师

上架建议：区块链

ISBN 978-7-121-35007-8



9 787121 350078 >

定价：79.00元



策划编辑：董  
责任编辑：葛  
封面设计：李

英  
娜  
玲

欢迎投稿  
dongying@phei.com.cn